Adaptive Oblivious Transfer and Generalization

Olivier Blazy, Céline Chevalier, Paul Germouty

Adaptive Oblivious Transfer and Generalization

Olivier Blazy[1], Céline Chevalier[2], and Paul Germouty[1]

[1] Université de Limoges, XLim, France
[2] Université Panthéon-Assas, Paris, France

Abstract. Oblivious Transfer (OT) protocols were introduced in the seminal paper of Rabin, and allow a user to retrieve a given number of lines (usually one) in a database, without revealing which ones to the server. The server is ensured that only this given number of lines can be accessed per interaction, and so the others are protected; while the user is ensured that the server does not learn the numbers of the lines required. This primitive has a huge interest in practice, for example in secure multi-party computation, and directly echoes to Symmetrically Private Information Retrieval (SPIR).

Recent Oblivious Transfer instantiations secure in the UC framework suffer from a drastic fallback. After the first query, there is no improvement on the global scheme complexity and so subsequent queries each have a global complexity of $\mathcal{O}(|DB|)$ meaning that there is no gain compared to running completely independent queries. In this paper, we propose a new protocol solving this issue, and allowing to have subsequent queries with a complexity of $\mathcal{O}(\log(|DB|))$, and prove the protocol security in the UC framework with adaptive corruptions and reliable erasures.

As a second contribution, we show that the techniques we use for Oblivious Transfer can be generalized to a new framework we call *Oblivious Language-Based Envelope* (OLBE). It is of practical interest since it seems more and more unrealistic to consider a database with uncontrolled access in access control scenarii. Our approach generalizes Oblivious Signature-Based Envelope, to handle more expressive credentials and requests from the user. Naturally, OLBE encompasses both OT and OSBE, but it also allows to achieve Oblivious Transfer with fine grain access over each line. For example, a user can access a line if and only if he possesses a certificate granting him access to such line.

We show how to generically and efficiently instantiate such primitive, and prove them secure in the Universal Composability framework, with adaptive corruptions assuming reliable erasures. We provide the new UC ideal functionalities when needed, or we show that the existing ones fit in our new framework.

The security of such designs allows to preserve both the secrecy of the database values and the user credentials. This symmetry allows to view our new approach as a generalization of the notion of Symmetrically PIR.

Keywords. Adaptive Oblivious Transfer, Oblivious Signature-Based Envelope, UC Framework, Private Information Retrieval.

1 Introduction

Oblivious Transfer (OT) is a notion introduced by Rabin in [Rab81]. In its classical 1-out-of-n version, it allows a user $\mathcal{U}$ to access a single line of a database while interacting with the server $\mathcal{S}$ owning the database. The user should be oblivious to the other line values, while the server should be oblivious to which line was indeed received. Oblivious transfer has a fundamental role for achieving secure multi-party computation: It is for example needed for every bit of input in Yao's protocol [Yao86] as well as for Oblivious RAM ([WHC+14] for instance), for every AND gate in the Boolean circuit computing the function in [GMW87] or for almost all known garbled circuits [BHR12].

Private Information Retrieval (PIR) schemes [CGKS95] allow a user to retrieve information from a database, while ensuring that the database does not learn which data were retrieved. With the increasing need for user privacy, these schemes are quite useful in practice, be they used for accessing records for email repositories, collection of webpages, music... But while protecting the privacy of the user, it is equally important that the user should not learn more information than he is allowed to. This is called database privacy and the corresponding protocol is called a Symmetrically Private Information Retrieval (SPIR), which is employed in practice, for medical data or biometric information for instance. This notion is closely related to Oblivious Transfer.

Due to their huge interest in practice, it is important to achieve low communication on these Oblivious Transfer protocols. A usual drawback is that the server usually has to send a message equivalent to the whole database each time the user requests a line. If it is logical that an OT protocol requires a cost linear in the size of the database for the first line queried, one may hope to amortize the cost for further queries between the same server and the same user (or even another user, if possible), reducing the efficiency gap between Private Information Retrieval schemes and their stronger equivalent Oblivious Transfer schemes. We thus deal in this paper with a more efficient way, which is to achieve *Adaptive* Oblivious Transfer, in which the user can adaptively ask several lines of the database. In such schemes, the server only sends his database once at the beginning of the protocol, and all the subsequent communication is in $o(n)$, more precisely logarithmic.

Smooth Projective Hash Functions (SPHF), used in conjunction with *Commitments* have become the classical way to deal with such secret message transfers. In a commitment scheme, the sender is going to commit to the line required (*i.e.* to give the receiver an analogue of a sealed envelope containing his value i) in such a way that he should not be able to open to a value different from the one he committed to (*binding* property), and that the receiver cannot learn anything about i (*hiding* property) before a potential opening phase. During the opening phase, however, the committer would be asked to reveal i in such a way that the receiver can verify it was indeed i that was contained in the envelope.

But, in our applications, there cannot be an opening phase, due to the *oblivious* requirements on the protocols and the secrecy of the database line i sent. The decommitment (opening phase) will thus be implicit, which means that the

committer does not really open its commitment, but rather convinces the receiver that it actually committed to the value it pretended to. We achieve this property thanks to *Smooth Projective Hash Functions* [CS02,GL03], which have been widely used in such circumstances (see [ACP09,KV11,BBC+13b,ABB+13,BC15] for instance). These hash functions are defined in such a way that their value can be computed in two different ways if the input belongs to a particular subset (the *language*), either using a private hashing key or a public projection key along with a private witness ensuring that the input belongs to the language. The hash value obtained is indistinguishable from random in case the input does not belong to the language (*smoothness*) and in case the input does belong to the language but no witness is known (*pseudo-randomness*).

In a nutshell, to ensure implicit decommitment, the sender will thus simply mask the database line with this hash value computed using the private hashing key. He will then send it along with the public projection key to the user, who will be able to compute the same hash value thanks to the randomness of the commitment of this line he sent in the first place (the randomness is the witness of the membership of the commitment to the language of commitments of this specific line). In order to ensure adaptive security in the universal composability framework, the commitments used are usually required to be both *extractable* (meaning that a simulator can recover the value i committed to thanks to a trapdoor) and *equivocable* (meaning that a simulator can open a commitment to a value i' different from the value i it committed to thanks to a trapdoor).

In order to simplify these commitments, which can be quite technical, we choose here to rely on words on more complex languages rather than on simple line numbers. More precisely, the user will first compute an equivocable commitment on the line number required, which will be his word w in the language. This word will then be encrypted under a CCA encryption scheme, and the SPHF will be constructed for this word (rather than for the line number), which will be simpler. Furthermore, this abstraction consisting in encoding line numbers as words in more complex languages will reveal useful in more general contexts, not only Oblivious Transfer, the simplest of which being Oblivious Signature Based Envelope.

Oblivious Signature-Based Envelope (OSBE) was introduced by Li, Du and Boneh in [LDB03]. OSBE schemes consider the case where Alice (the receiver) is a member of an organization and possesses a certificate produced by an authority attesting she actually belongs to this organization. Bob (the sender) wants to send a private message P to members of this organization. However due to the sensitive nature of the organization, Alice does not want to give Bob neither her certificate nor a proof she belongs to the organization. OSBE lets Bob send an obfuscated version of this message P to Alice, in such a way that she will be able to find P if and only if she is in the required organization. In the process, Bob cannot decide whether Alice does really belong to the organization. We even manage to construct a more general framework to capture many protocols around trust negotiation, where the user receives a message if and only if he

possesses some credentials or specific accreditations. As a reference to OSBE, we call this framework *Oblivious Language-Based Envelope* (OLBE).

Related Work. Since the original paper [Rab81], several instantiations and optimizations of OT protocols have appeared in the literature, including proposals in the UC framework [NP01, CLOS02]. More recently, new instantiations have been proposed, trying to reach round-optimality [HK07], or low communication costs [PVW08]. Recent schemes like [ABB+13, BC15] manage to achieve round-optimality while maintaining a small communication cost. Choi *et al.* [CKWZ13] also propose a generic method and an efficient instantiation secure against adaptive corruptions in the CRS model with erasures, but it is only 1-out-of-2 and it does not scale to 1-out-of-n OT, for $n > 2$. As far as adaptive versions of those protocols are concerned, this problem was first studied by [NP97, GH07, KNP11], and more recently UC secure instantiations were proposed, but unfortunately either under the Random Oracle, or under not so classical assumptions such as q-Hidden LRSW or later on q-SDH [GH08, JL09, RKP09, CDH12, GD14], but without allowing adaptive corruptions.

Concerning automated trust negotiation, two frameworks have been proposed to encompass the symmetric protocols (Password-based Authenticated Key-Exchange, Secret Handshakes and Verfier-Based PAKE): The Credential Authenticated Key Exchange [CCGS10], and Language-based Authenticated Key Exchange (LAKE) [BBC+13a], in which two parties establish a common session key if and only if they hold credentials that belong to specific (and possibly independent) languages chosen by the other party. As for OSBE, the authors in [BPV12] improved the security model initially proposed in [LDB03], showing how to use Smooth Projective Hash Functions to do implicit proof of knowledge, and proposed the first efficient instantiation of OSBE, under a standard hypothesis. It fits, as well as Access Controlled Oblivious Transfer [CDN09, CDNZ11], Priced Oblivious Transfer [AIR01, RKP09]) and Conditional Oblivious Transfer [DOR99], into the generic notion of Conditional Disclosure of Secrets (see for instance [AIR01, GIKM98, BGN05, LL07, Wee14, IW14, Att14, GKW15]).

Contributions. Our first contribution is to give the first adaptive Oblivious Transfer protocol secure in the UC framework with adaptive corruptions under classical assumptions (MDDH) and assuming reliable erasures. We show how to instantiate the needed building blocks using classical assumptions, using or extending various basic primitives in order to fit the MDDH framework introduced in [EHK+13]. In our scheme, the server first preprocesses its database in a time linear in the length of the database and transfers it to the receiver. After that, the receiver and the sender can run many instances of the protocol on the same database as input and adaptively chosen inputs from the receiver, with a cost sublinear in the database.

It is interesting to note that our resulting adaptive Oblivious Transfer scheme has an amortized complexity in $\mathcal{O}(\log |DB|)$, which is similar to current Private Information Retrieval instantiations [KLL+15], that have weaker security prerequisites, and much better than current UC secure Oblivious Transfer under classical assumptions (as they are in $\mathcal{O}(|DB|)$. Compared to existing versions

cited above (either proven in classical security models, or in the UC framework but only with static corruptions and under non classical ssumptions), we manage to prove its security under classical assumptions, like SXDH, and allow UC security with adaptive user corruptions.

As a side result, it is worth noting that we follow some ideas developed in the construction explained in [GH07] around Blind Identity-Based Encryption and provide techniques in order to transform IBE schemes into blind ones, applying them to revisit the one given in [BKP14], in order to show how we can answer blind user secret key-retrieval, which can be of independent interest.

As a second contribution, we propose our new notion, that we call Oblivious Language-Based Envelope. We provide a security model by giving a UC ideal functionality, and show that this notion supersedes the classical asymmetric automated trust negotiation schemes recalled above such as Oblivious Transfer and Oblivious Signature-Based Envelope. We show how to choose the languages in order to obtain from our framework all the corresponding ideal functionalities, recovering the known ones (such as OT) and providing the new ones (such as OSBE, to the best of our knowledge). We then give a generic construction scheme fulfilling our ideal functionality, which directly gives generic constructions for the specific cases (OT, OSBE). Finally, we show how to instantiate the different simple building blocks in order to recover the classical efficient instantiations of these schemes from our framework. In addition to the two cases most studied (OT, OSBE), we also propose what we call *Conditioned Oblivious Transfer*, which encompasses Access Controlled Oblivious Transfer, Priced Oblivious Transfer and Conditional Oblivious Transfer, and in which the access to each line of the database is hidden behind some possibly secret restriction, be it a credential, a price, or an access policy. The advantage of the OLBE framework on the notion of Conditional Disclosure of Secrets is to allow generic constructions of a large subclass of schemes, as long as two participants are involved. It can be easily applied to any language expressing some new access control policy. Furthermore, those instantiations fit into a global security model, allowing to uniformize (for the better) the security expectations for such schemes. In particular, we allow security in the UC framework with adaptive corruptions for all our constructions (which was already known for some primitives cited above, but not all), and manage to achieve this level of security while staying in the standard model with standard hypothesis.

2 Definitions and Building Blocks

2.1 Notations for Classical Primitives

Throughout this paper, we use the notation $\mathfrak{K}$ for the security parameter.

Digital Signature. A digital signature scheme $\mathcal{S}$ [DH76,GMR88] allows a signer to produce a verifiable proof that he indeed produced a message. It is described through four algorithms $\sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. The formal definitions are given in Appendix A of the additional content.

Encryption. An encryption scheme $\mathcal{C}$ is described through four algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$. The formal definitions are given in Appendix A of the additional content.

Commitment and Chameleon Hash. Commitments allow a user to commit to a value without revealing it, but without the possibility to later change his mind. It is composed of four algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Decommit})$. Informally, it is extractable if a simulator knowing a certain trapdoor can recover the value committed to, and it is equivocable if a simulator, knowing another trapdoor, can open the commitment to another value than the one it actually committed to. This directly echoes to Chameleon Hashes, traditionally defined by three algorithms $\mathsf{CH} = (\mathsf{KeyGen}, \mathsf{CH}, \mathsf{Coll})$. The formal definitions are given in Appendix A of the additional content.

2.2 Identity-Based Encryption, Identity-based Key Encapsulation

Identity Based encryption was first introduced by Shamir in [Sha84] who was expecting an encryption scheme where no public key will be needed for sending a message to a precise user, defined by his identity. Thus any user wanting to send a private message to a user only need this user's identity and a master public key. It took 17 years for the cryptographic community to find a way to realize this idea. The first instantiation was proposed in [BF01] by Boneh and Franklin. It can be described as an identity-based key encapsulation (IBKEM) scheme IBKEM which consists of four algorithms $\mathsf{IBKEM} = (\mathsf{Gen}, \mathsf{USKGen}, \mathsf{Enc}, \mathsf{Dec})$. Every IBKEM can be transformed into an ID-based encryption scheme IBE using a (one-time secure) symmetric cipher.

Definition 1 (Identity-based Key Encapsulation Scheme). *An identity-based key encapsulation scheme* IBKEM *consists of four PPT algorithms* $\mathsf{IBKEM} = (\mathsf{Gen}, \mathsf{USKGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following properties.*

- $\mathsf{Gen}(\mathfrak{K})$*: returns the (master) public/secret key* $(\mathsf{mpk}, \mathsf{msk})$*. We assume that* mpk *implicitly defines an identity space* $\mathcal{ID}$*, a key space* $\mathcal{KS}$*, and ciphertext space* CS*.*
- $\mathsf{USKGen}(\mathsf{msk}, \mathsf{id})$*: returns the user secret-key* $\mathsf{usk}[\mathsf{id}]$ *for identity* $\mathsf{id} \in \mathcal{ID}$*.*
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id})$*: returns the symmetric key* $\mathsf{K} \in \mathcal{KS}$ *together with a ciphertext* $\mathsf{C} \in \mathsf{CS}$ *with respect to identity* id*.*
- $\mathsf{Dec}(\mathsf{usk}[\mathsf{id}], \mathsf{id}, \mathsf{C})$*: returns the decapsulated key* $\mathsf{K} \in \mathcal{K}$ *or the reject symbol* $\perp$*.*

For perfect correctness we require that for all $\mathfrak{K} \in \mathbb{N}$*, all pairs* $(\mathsf{mpk}, \mathsf{msk})$ *generated by* $\mathsf{Gen}(\mathfrak{K})$*, all identities* $\mathsf{id} \in \mathcal{ID}$*, all* $\mathsf{usk}[\mathsf{id}]$ *generated by* $\mathsf{USKGen}(\mathsf{msk}, \mathsf{id})$ *and all* (K, C) *output by* $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id})$*:* $\Pr[\mathsf{Dec}(\mathsf{usk}[\mathsf{id}], \mathsf{id}, \mathsf{C}) = \mathsf{K}] = 1$.

The security requirements for an IBKEM we consider here are indistinguishability and anonymity against chosen plaintext and identity attacks ($\mathsf{IND\text{-}ID\text{-}CPA}$ and $\mathsf{ANON\text{-}ID\text{-}CPA}$). Instead of defining both security notions separately, we define pseudorandom ciphertexts against chosen plaintext and identity attacks ($\mathsf{PR\text{-}ID\text{-}CPA}$) which means that challenge key and ciphertext are both pseudorandom. Note that $\mathsf{PR\text{-}ID\text{-}CPA}$ trivially implies $\mathsf{IND\text{-}ID\text{-}CPA}$ and $\mathsf{ANON\text{-}ID\text{-}CPA}$. We define $\mathsf{PR\text{-}ID\text{-}CPA}$-security of IBKEM formally via the games given in Figure 1.

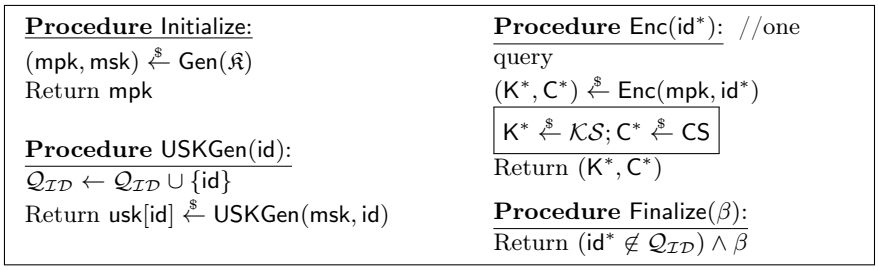


Fig. 1. PR-ID-CPA-security: Security Games $\mathsf{PR\text{-}ID\text{-}CPA}_{\mathsf{real}}$ and $\mathsf{PR\text{-}ID\text{-}CPA}_{\mathsf{rand}}$ (boxed).

Definition 2 (PR-ID-CPA Security). *An ID-based key encapsulation scheme* IBKEM *is* PR-ID-CPA*-secure if for all PPT $\mathscr{A}$, the following advantage is negligible:* $\mathsf{Adv}^{\mathsf{pr\text{-}id\text{-}cpa}}_{\mathsf{IBKEM}}(\mathscr{A}) = |\Pr[\mathsf{PR\text{-}ID\text{-}CPA}^{\mathscr{A}} = 1] - \Pr[\mathsf{PR\text{-}ID\text{-}CPA}_{\mathsf{rand}}{}^{\mathscr{A}} \Rightarrow 1]|$.

2.3 Smooth Projective Hashing and Languages

Smooth projective hash functions (SPHF) were introduced by Cramer and Shoup in [CS02] for constructing encryption schemes. A projective hashing family is a family of hash functions that can be evaluated in two ways: using the (secret) hashing key, one can compute the function on every point in its domain, whereas using the (public) *projected* key one can only compute the function on a special subset of its domain. Such a family is deemed *smooth* if the value of the hash function on any point outside the special subset is independent of the projected key. The notion of SPHF has already found applications in various contexts in cryptography (*e.g.* [GL03, Kal05, ACP09]). A Smooth Projective Hash Function over a language $\mathfrak{L} \subset X$, onto a set $\mathcal{G}$, is defined by five algorithms (Setup, HashKG, ProjKG, Hash, ProjHash):

- $\mathsf{Setup}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and the description of an $\mathcal{NP}$ language $\mathfrak{L}$;
- $\mathsf{HashKG}(\mathfrak{L}, \mathsf{param})$, outputs a hashing key hk for the language $\mathfrak{L}$;
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, derives the projection key hp from hk.
- $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, outputs a hash value $v \in \mathcal{G}$, thanks to hk and W.
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w)$, outputs the hash value $v' \in \mathcal{G}$, thanks to the projection key hp and the witness w that $W \in \mathfrak{L}$.

In the following, we consider $\mathfrak{L}$ as a hard-partitioned subset of X, *i.e.* it is computationally hard to distinguish a random element in $\mathfrak{L}$ from a random element in $X \setminus \mathfrak{L}$. An SPHF should satisfy the following properties:

- *Correctness*: Let $W \in \mathfrak{L}$ and w a witness of this membership. Then, for all hashing keys hk and associated projection keys hp we have
$$\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W) = \mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w).$$
- *Smoothness*: For all $W \in X \setminus \mathfrak{L}$ the following distributions are statistically indistinguishable:

$$\Delta_0 = \left\{ (\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \mathsf{hk} = \mathsf{HashKG}(\mathfrak{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W), \\ v = \mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W) \end{array} \right\}$$

$$\Delta_1 = \left\{ (\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \mathsf{hk} = \mathsf{HashKG}(\mathfrak{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W), v \xleftarrow{\$} \mathcal{G} \end{array} \right\}.$$

This is formalized by: $\mathsf{Adv}^{\mathsf{smooth}}_{\mathsf{SPHF}}(\mathfrak{K}) = \sum_{V \in \mathbb{G}} |\Pr_{\Delta_1}[v = V] - \Pr_{\Delta_0}[v = V]|$ is negligible.

– *Pseudo-Randomness*: If $W \in \mathfrak{L}$, then without a witness of membership the two previous distributions should remain computationally indistinguishable. For any adversary $\mathcal{A}$ within reasonable time, this advantage is negligible:

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathsf{SPHF},\mathcal{A}}(\mathfrak{K}) = |\Pr_{\Delta_1}[\mathcal{A}(\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) = 1] - \Pr_{\Delta_0}[\mathcal{A}(\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) = 1]|$$

Languages. The language $\mathfrak{L} \subset X$ used in our definition should be a hard-partitioned subset of X, *i.e.* it is computationally hard to distinguish a random element in $\mathfrak{L}$ from a random element not in $\mathfrak{L}$ (see formal definition in [GL03, AP06]). Furthermore, the language should fulfill the following properties:

– *Publicly Verifiable:* Given a word x in X, anyone should be able to decide in polynomial time whether $x \in \mathfrak{L}$ or not.

– *Self-Randomizable*: Given a word in the language, anyone should be able to sample a new word in the language[1], and the distribution of this resampling should be indistinguishable from an honest distribution. This will be used in order to prevent an adversary, or the authority in charge of distributing the words, to learn which specific form of the word was used by the user.

In case we consider several languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$, we also assume it is a *Trapdoor Collection of Languages*: It is computationally hard to sample an element in $\mathfrak{L}_1 \cap \cdots \cap \mathfrak{L}_n$, except if one possesses a trapdoor tk (without the knowledge of the potential secret keys)[2]. For instance, if for all i, $\mathfrak{L}_i$ is the language of the equivocable commitments on words in an inner language $\widetilde{\mathfrak{L}}_i = \{i\}$ (as we will consider for OT), the common trapdoor key can be the equivocation trapdoor.

Depending on the applications, we can assume a *Keyed Language*, which means that it is set by a trusted authority, and that it is hard to sample fresh elements from scratch in the language without the knowledge of a secret language key $\mathsf{sk}_{\mathfrak{L}}$. In this case, the authority is also in charge of giving a word in the language to the receiver.

In case the language is keyed, we assume it is also a *Trapdoor Language*: We assume the existence of a trapdoor tk_L allowing a simulator to sample an element in $\mathfrak{L}$ (without the knowledge of the potential secret key $\mathsf{sk}_{\mathfrak{L}}$). For instance, for a language of valid Waters signatures of a message M (as we will consider for OSBE), one can think of $\mathsf{sk}_{\mathfrak{L}}$ as being the signing key, whereas the trapdoor $\mathsf{tk}_{\mathfrak{L}}$ can be the discrete logarithm of h in basis g.[3]

[1] It should be noted that this property is not incompatible with the potential secret key of the language in case it is keyed (see below).

[2] This implicitly means that the languages are compatible, in the sense that one can indeed find a word belonging to all of them.

[3] As another example, one may think of more expressive languages which may not rely directly on generators fixed by the CRS. In this case, one can assume that the CRS contains parameters for an encryption and an associated NIZK proof system. The description of such a language is thus supplemented with an encryption of the lan-

2.4 Security Assumptions

Due to lack of space, instantiations of the primitives recalled above are given in Appendix B of the additional content and we only give here the security assumptions.

Security Assumption: Pairing groups and Matrix Diffie-Hellman Assumption. Let GGen be a probabilistic polynomial time (PPT) algorithm that on input $1^{\mathfrak{K}}$ returns a description $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ of asymmetric pairing groups where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are cyclic groups of order p for a $\mathfrak{K}$-bit prime p, g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2$ is an efficiently computable (non-degenerated) bilinear map. Define $g_T := e(g_1, g_2)$, which is a generator in $\mathbb{G}_T$.

We use implicit representation of group elements as introduced in [EHK+13]. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$ define $[a]_s = g_s^a \in \mathbb{G}_s$ as the *implicit representation* of a in $\mathbb{G}_s$. More generally, for a matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{Z}_p^{n \times m}$ we define $[\boldsymbol{A}]_s$ as the implicit representation of $\boldsymbol{A}$ in $\mathbb{G}_s$:

$$[\boldsymbol{A}]_s := \begin{pmatrix} g_s^{a_{11}} & \dots & g_s^{a_{1m}} \\ & & \\ g_s^{a_{n1}} & \dots & g_s^{a_{nm}} \end{pmatrix} \in \mathbb{G}_s^{n \times m}$$

We will always use this implicit notation of elements in $\mathbb{G}_s$, i.e., we let $[a]_s \in \mathbb{G}_s$ be an element in $\mathbb{G}_s$. Note that from $[a]_s \in \mathbb{G}_s$ it is generally hard to compute the value a (discrete logarithm problem in $\mathbb{G}_s$). Further, from $[b]_T \in \mathbb{G}_T$ it is hard to compute the value $[b]_1 \in \mathbb{G}_1$ and $[b]_2 \in \mathbb{G}_2$ (pairing inversion problem). Obviously, given $[a]_s \in \mathbb{G}_s$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax]_s \in \mathbb{G}_s$. Further, given $[a]_1, [b]_2$ one can efficiently compute $[ab]_T$ using the pairing e. For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^k$ define $e([\boldsymbol{a}]_1, [\boldsymbol{b}]_2) := [\boldsymbol{a}^\top \boldsymbol{b}]_T \in \mathbb{G}_T$.

We recall the definition of the matrix Diffie-Hellman (MDDH) assumption [EHK+13].

Definition 3 (Matrix Distribution). *Let $k \in \mathbb{N}$. We call $\mathcal{D}_k$ a matrix distribution if it outputs matrices in $\mathbb{Z}_p^{(k+1) \times k}$ of full rank k in polynomial time.*

Without loss of generality, we assume the first k rows of $\boldsymbol{A} \xleftarrow{\$} \mathcal{D}_k$ form an invertible matrix, we denote this matrix $\overline{\boldsymbol{A}}$, while the last line is denoted $\underline{A}$. The $\mathcal{D}_k$-Matrix Diffie-Hellman problem is to distinguish the two distributions $([\boldsymbol{A}], [\boldsymbol{A}\boldsymbol{w}])$ and $([\boldsymbol{A}], [\boldsymbol{u}])$ where $\boldsymbol{A} \xleftarrow{\$} \mathcal{D}_k$, $\boldsymbol{w} \xleftarrow{\$} \mathbb{Z}_p^k$ and $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_p^{k+1}$.

Definition 4 ($\mathcal{D}_k$-Matrix Diffie-Hellman Assumption $\mathcal{D}_k$-MDDH). *Let $\mathcal{D}_k$ be a matrix distribution and $s \in \{1, 2, T\}$. We say that the $\mathcal{D}_k$-Matrix Diffie-Hellman ($\mathcal{D}_k$-MDDH) Assumption holds relative to GGen in group $\mathbb{G}_s$ if for all PPT adversaries $\mathcal{D}$,*

$$\mathbf{Adv}_{\mathcal{D}_k, \mathsf{GGen}}(\mathcal{D}) := |\Pr[\mathcal{D}(\mathcal{G}, [\boldsymbol{A}]_s, [\boldsymbol{A}\boldsymbol{w}]_s) = 1] - \Pr[\mathcal{D}(\mathcal{G}, [\boldsymbol{A}]_s, [\boldsymbol{u}]_s) = 1]| = \mathsf{negl}(\lambda),$$

where the probability is taken over $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, $\boldsymbol{A} \xleftarrow{\$} \mathcal{D}_k, \boldsymbol{w} \xleftarrow{\$} \mathbb{Z}_p^k, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_p^{k+1}$.

guage trapdoor, and a non-interactive zero-knowledge proof that the encrypted value is indeed a trapdoor for the said language. Using the knowledge of the decryption key, the simulator is able to recover the trapdoor.

For each $k \geq 1$, [EHK+13] specifies distributions $\mathcal{L}_k$, $\mathcal{U}_k$, ... such that the corresponding $\mathcal{D}_k$-MDDH assumption is the k-Linear assumption, the k-uniform and others. All assumptions are generically secure in bilinear groups and form a hierarchy of increasingly weaker assumptions. The distributions are exemplified for $k = 2$, where $a_1, \ldots, a_6 \xleftarrow{\$} \mathbb{Z}_p$.

$$\mathcal{L}_2 : \boldsymbol{A} = \begin{pmatrix} a_1 & 0 \\ 0 & a_2 \\ 1 & 1 \end{pmatrix} \quad \mathcal{U}_2 : \boldsymbol{A} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \\ a_5 & a_6 \end{pmatrix}.$$

It was also shown in [EHK+13] that $\mathcal{U}_k$-MDDH is implied by all other $\mathcal{D}_k$-MDDH assumptions.

Lemma 5 (Random self reducibility [EHK+13]). *For any matrix distribution $\mathcal{D}_k$, $\mathcal{D}_k$-MDDH is random self-reducible. In particular, for any $m \geq 1$,*

$$\mathbf{Adv}_{\mathcal{D}_k,\mathsf{GGen}}(\mathcal{D}) + \tfrac{1}{q-1} \geq \mathbf{Adv}^m_{\mathcal{D}_k,\mathsf{GGen}}(\mathcal{D}')$$

where $\mathbf{Adv}^m_{\mathcal{D}_k,\mathsf{GGen}}(\mathcal{D}') := \Pr[\mathcal{D}'(\mathcal{G}, [\boldsymbol{A}], [\boldsymbol{AW}]) \Rightarrow 1] - \Pr[\mathcal{D}'(\mathcal{G}, [\boldsymbol{A}], [\boldsymbol{U}]) \Rightarrow 1]$, *with* $\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, $\boldsymbol{A} \xleftarrow{\$} \mathcal{D}_k, \boldsymbol{W} \xleftarrow{\$} \mathbb{Z}_p^{k \times m}, \boldsymbol{U} \xleftarrow{\$} \mathbb{Z}_p^{(k+1)\times m}$.

Remark: It should be noted that $\mathcal{L}_1, \mathcal{L}_2$ are respectively the SXDH and DLin assumptions.

2.5 Security Models

UC Framework. The goal of the UC framework [Can01] is to ensure that UC-secure protocols will continue to behave in the ideal way even if executed in a concurrent way in arbitrary environments. It is a simulation-based model, relying on the indistinguishability between the real world and the ideal world. In the ideal world, the security is provided by an ideal functionality $\mathcal{F}$, capturing all the properties required for the protocol and all the means of the adversary. In order to prove that a protocol Π emulates $\mathcal{F}$, one has to construct, for any polynomial adversary $\mathscr{A}$ (which controls the communication between the players), a simulator $\mathscr{S}$ such that no polynomial environment $\mathcal{Z}$ can distinguish between the real world (with the real players interacting with themselves and $\mathscr{A}$ and executing the protocol π) and the ideal world (with dummy players interacting with $\mathscr{S}$ and $\mathcal{F}$) with a significant advantage. The adversary can be either *adaptive*, *i.e.* allowed to corrupt users whenever it likes to, or *static*, *i.e.* required to choose which users to corrupt prior to the execution of the session sid of the protocol. After corrupting a player, $\mathscr{A}$ has complete access to the internal state and private values of the player, takes its entire control, and plays on its behalf.

Simple UC Framework. Canetti, Cohen and Lindell formalized a simpler variant in [CCL15], that we use here. This simplifies the description of the functionalities for the following reasons (in a nutshell): All channels are automatically assumed to be authenticated (as if we worked in the $\mathcal{F}_{\text{AUTH}}$-hybrid model); There is no need for *public delayed outputs* (waiting for the adversary before delivering a message to a party), neither for an explicit description of the corruptins. We refer the interested reader to [CCL15] for details.

3 UC-secure Adaptive Oblivious Transfer

As explained in the introduction, the classical OT constructions based on the commitment/SPHF paradigm (among the latest in the UC framework, [CKWZ13, ABB+13,BC15]) require the server to send an encryption of the complete database for each line required by the user (thus $O(n)$ each time). We here give a protocol requiring $O(\log(n))$ for each line (except the first one, still in $O(n)$), in the UC framework with adaptive corruptions under classical assumptions (MDDH).

3.1 Definition and Security Model

Using implicit decommitment in the UC framework implies a very strong commitment primitive (formalized as SPHF-friendly commitments in [ABB+13]), which is both extractable and equivocable. Our idea is here to split these two properties by using on the one hand an equivocable commitment and on the other hand an (extractable) CCA encryption scheme by generalizing the way to access a line in the database. Indeed, we suggest not to consider anymore the line numbers as numbers in $\{1,\dots,n\}$ but rather to "encode" them (the exact encoding will depend on the protocol): For every line i, a word W_i in the language $\mathfrak{L}_i$ will correspond to a representation of line i. This representation must be publicly verifiable, in the sense that anyone can associate i to a word W_i. We formalize this in the following definition of oblivious transfer[4]. In such a protocol, a server $\mathcal{S}$ possesses a database of n lines $(P_1,\dots,P_n) \in (\{0,1\}^{\mathfrak{K}})^n$. A user $\mathcal{U}$ will be able to recover P_i (in an oblivious way) as soon as he owns a word $W_i \in \mathfrak{L}_i$. The languages $(\mathfrak{L}_1,\dots,\mathfrak{L}_n)$ will be assumed to be a trapdoor collection of languages, publicly verifiable and self-randomizable. As we consider simulation-based security (in the UC framework), we allow a simulated setup SetupT to be run instead of the classical setup Setup in order to allow the simulator to possess some trapdoors. Those two setup algorithms should be indistinguishable.

Definition 6 (Oblivious Transfer). *An* OT *scheme is defined by five algorithms* $(\mathsf{Setup},\mathsf{KeyGen},\mathsf{DBGen},\mathsf{Samp},\mathsf{Verify})$*, along with an interactive protocol* $\mathsf{Protocol}\langle\mathcal{S},\mathcal{U}\rangle$*:*

- $\mathsf{Setup}(1^{\mathfrak{K}})$*, where* $\mathfrak{K}$ *is the security parameter, generates the global parameters* param*, among which the number* n*;*

or $\mathsf{SetupT}(1^{\mathfrak{K}})$*, where* $\mathfrak{K}$ *is the security parameter, additionally allows the existence*[5] *of a trapdoor* tk *for the collection of languages* $(\mathfrak{L}_1,\dots,\mathfrak{L}_n)$*.*

- $\mathsf{KeyGen}(\mathsf{param},\mathfrak{K})$ *generates, for all* $i \in \{1,\dots,n\}$*, the description of the language* $\mathfrak{L}_i$ *(as well as the language key* $\mathsf{sk}_{\mathfrak{L}_i}$ *if need be). If the parameters* param *were defined by* SetupT*, this implicitly also defines the common trapdoor* tk *for the collection of languages* $(\mathfrak{L}_1,\dots,\mathfrak{L}_n)$*.*

[4] The adaptive version only implies that the database $(P_1,\dots,P_n)$ is sent only once in the interaction, while the user can query several lines (*i.e.* several words), in an adaptive way.

[5] The specific trapdoor will depend on the languages and be computed in the KeyGen algorithm.

The functionality $\mathcal{F}_{\mathrm{OT}}^{\mathfrak{L}}$ is parametrized by a security parameter $\mathfrak{K}$ and a set of languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ along with the corresponding public verification algorithms $(\mathsf{Verify}_1, \ldots, \mathsf{Verify}_n)$. It interacts with an adversary $\mathscr{S}$ and a set of parties $\mathfrak{P}_1, \ldots, \mathfrak{P}_N$ via the following queries:

- **Upon receiving an input $(\mathtt{NewDataBase}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1, \ldots, P_n))$ from party $\mathfrak{P}_i$**, with $P_i \in \{0,1\}^{\mathfrak{K}}$ for all i: record the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1, \ldots, P_n))$ and reveal $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$. Ignore further $\mathtt{NewDataBase}$-message with the same ssid from $\mathfrak{P}_i$.
- **Upon receiving an input $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, W_i)$ from party $\mathfrak{P}_j$**: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1, \ldots, P_n))$ is not recorded. Otherwise, reveal $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$ and send $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, P_i')$ to $\mathfrak{P}_j$ where $P_i' = P_i$ if $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ returns 1, and $P_i' = \perp$ otherwise.
  (Non-Adaptive case: Ignore further $\mathtt{Receive}$-message with the same ssid from $\mathfrak{P}_j$.)

Fig. 2. Ideal Functionality for (Adaptive) Oblivious Transfer $\boldsymbol{\mathcal{F}}_{\mathrm{OT}}^{\mathfrak{L}}$

- $\mathsf{Samp}(\mathsf{param})$ *or* $\mathsf{Samp}(\mathsf{param}, (\mathsf{sk}_{\mathfrak{L}_i})_{i\in\{1,\ldots,n\}})$ *generates a word* $W_i \in \mathfrak{L}_i$*;*
- $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ *checks whether* W_i *is a valid word in the language* $\mathfrak{L}_i$*. It outputs* 1 *if the word is valid,* 0 *otherwise;*
- $\mathsf{Protocol}\langle \mathcal{S}((\mathfrak{L}_1, \ldots, \mathfrak{L}_n), (P_1, \ldots, P_n)), \mathcal{U}((\mathfrak{L}_1, \ldots, \mathfrak{L}_n), W_i)\rangle$ *between the server* $\mathcal{S}$ *with the private database* $(P_1, \ldots, P_n)$ *and corresponding languages* $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$*, and the user* $\mathcal{U}$ *with the same languages and the word* W_i*, proceeds as follows. If the algorithm* $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ *returns* 1*, then* $\mathcal{U}$ *receives* P_i*, otherwise it does not. In any case,* $\mathcal{S}$ *does not learn anything.*

The ideal functionality of an Oblivious Transfer (OT) protocol was given in [Can01, CKWZ13, ABB+13], and an adaptive version in [GH08]. We here combine them and rewrite it in simple UC and using our language formalism (instead of directly giving a number line s to the functionality, the user will give it a word $W_s \in \mathfrak{L}_s$). The resulting functionality $\mathcal{F}_{\mathrm{OT}}^{\mathfrak{L}}$ is given in Figure 2.

3.2 High Level Idea of the Construction

Our construction is inspired by [GH07], who propose a neat framework allowing to achieve adaptive Oblivious Transfer in the simulation model. Their construction is quite simple: It requires a *blind identity-based encryption*, in other words, an IBE scheme in which there is a way to query for a user key generation without the authority (here the server) learning the targeted identity (here the line in the database). Once such a Blind IBE is defined, one can conveniently obtain an oblivious transfer protocol by asking the database to encrypt (once and for all) each line for an identity (the i-th line being encrypted for the identity i), and having the user do a blind user key generation query in order to recover the key corresponding to the line he expects to learn.

This approach is round-optimal: After the database preparation, the first flow is sent by the user as a commitment to the identity, and the second one is sent by

the server with the blinded expected information (here masked by an SPHF). But several technicalities have arised because of the UC framework. For instance, we had to commit to words in specific languages (so as to ensure extractability and equivocability) as well as to *fragment* the IBE keys into bits in order to achieve $O(\log n)$ in both flows. This allows us to achieve the first UC-secure adaptive OT protocol allowing adaptive corruptions. More details follow.

3.3 Building Blocks: From an IBE to a Blind Fragmented IBE

Definition and Security Properties of a Blind IBE Scheme. Following [BKP14], we recalled in Section 2.2 page 6 the definitions, notations and security properties for an IBE scheme, seen as an identity-based key encapsulation (IBKEM) scheme. We continue to follow the KEM formalism by adapting the definition of a Blind IBE scheme given in [GH07] to this setting.

Definition 7 (Blind Identity-based Key Encapsulation Scheme). *A blind identity-based key encapsulation scheme* BlindIBKEM *consists of four PPT algorithms* $(\mathsf{Gen}, \mathsf{BlindUSKGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following properties.*

- Gen, Enc *and* Dec *are defined as for a traditional* IBKEM *scheme.*
- $\mathsf{BlindUSKGen}(\langle(\mathcal{S}, \mathsf{msk})(\mathcal{U}, \mathsf{id}, \ell; \boldsymbol{\rho})\rangle)$ *is an interactive protocol, in which an honest user* $\mathcal{U}$ *with identity* $\mathsf{id} \in \mathcal{ID}$ *obtains the corresponding user secret key* $\mathsf{usk}[\mathsf{id}]$ *from the master authority* $\mathcal{S}$ *or outputs an error message, while* $\mathcal{S}$*'s output is nothing or an error message (*ℓ *is a label and* $\boldsymbol{\rho}$ *the randomness).*

Defining the security of a BlindIBKEM requires two additional properties, stated as follows (see [GH07, pages 6 and 7] for the formal security games):

1. **Leak-free Secret Key Generation** (called Leak-free Extract for Blind IBE security in the original paper): A potentially malicious user cannot learn anything by executing the BlindUSKGen protocol with an honest authority which he could not have learned by executing the USKGen protocol with an honest authority; Moreover, as in USKGen, the user must know the identity for which he is extracting a key.
2. **Selective-failure Blindness**: A potentially malicious authority cannot learn anything about the user's choice of identity during the BlindUSKGen protocol; Moreover, the authority cannot cause the BlindUSKGen protocol to fail in a manner dependent on the user's choice.

   For our applications, we only need a weakened property for blindness:[6]
3. **Weak Blindness**: A potentially malicious authority cannot learn anything about the user's choice of identity during the BlindUSKGen protocol.

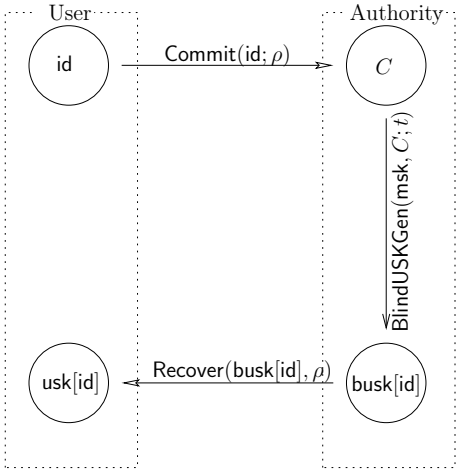
[6] Two things to note: First, Selective Failure would be considered as a Denial of Service in the Oblivious Transfer setting. Then, we do not restrict ourselves to schemes where the blindness adversary has access to the generated user keys, as reliable erasures in the OT protocol provide us a way to forget them before being corrupted (otherwise we would need to use a randomizable base IBE).

High-Level Idea of the Transformation. We now show how to obtain a BlindIBKEM scheme from any IBKEM scheme. From a high-level point of view, this transformation mixes two pre-existing approaches.

First, we are going to consider a reverse Naor transform [BF01, CFH+07]: He drew a parallel between Identity-Based Encryption schemes and signature schemes, by showing that a user secret key on an identity can be viewed as the signature on this identity, the verification process therefore being a test that any chosen valid ciphertext for the said identity can indeed be decrypted using the *signature* scheme.

Then, we are going to use Fischlin [Fis06] round-optimal approach to blind signatures, where the whole interaction is done in one pass: First, the user commits to the message, then he recovers a signature linked to his commitment. For sake of simplicity, instead of using a Non-Interactive Zero-Knowledge Proof of Knowledge of a signature, we are going to follow the [BFPV10, BPV12] approach, where thanks to an additional term, the user can extract a signature on the identity from a signature on the committed identity.

The main idea of the transformation of the IBKEM scheme in order to blind a user key request is described in Figure 3.



1. A user commits to the targeted identity id using some randomness ρ.
2. The authority possesses an algorithm allowing it to generate keys for committed identities using its master secret key msk, and some randomness t, in order to obtain a blinded user secret key busk[id].
3. The user using solely the randomness used in the initial commitment is able to recover the requested secret key from the authority's generated value.

Fig. 3. Generic Transformation of an IBE into a Blind IBE (naive approach)

Blinding a User Key Request via Implicit Decommitment. It now remains to explain how one can fulfill the idea highlighted in Figure 3. The technique uses a smooth projective hash function (see Section 2.3), and is often called "implicit decommitment" in recent works: the IBE secret key is sent hidden in such a way that it can only be recovered if the user knows how to open the

initial commitment on the correct identity. We assume the existence of a labeled CCA-encryption scheme $\mathcal{E} = (\mathsf{Setup}_{\mathrm{cca}}, \mathsf{KeyGen}_{\mathrm{cca}}, \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}, \mathsf{Decrypt}^{\ell}_{\mathrm{cca}})$ compatible with an SPHF defined by $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ onto a set G. By "compatible", we mean that the SPHF can be defined over a language $\mathfrak{L}_{C,\mathsf{id}} \subset X$, where $\mathfrak{L}_{C,\mathsf{id}} = \{C \mid \exists \boldsymbol{\rho}$ such that $C = \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(\mathsf{id}; \boldsymbol{\rho})\}$. In the KeyGen algorithm, the description of the language $\mathfrak{L}_{\mathsf{id}} = \{\mathsf{id}\}$ thus implicitly defines the language $\mathfrak{L}_{C,\mathsf{id}}$ of CCA-encryptions of elements of $\mathfrak{L}_{\mathsf{id}}$. We additionally use a key derivation function KDF to derive a pseudo-random bit-string $K \in \{0,1\}^{\mathfrak{K}}$ from a pseudo-random element $v \in G$. One can use the Leftover-Hash Lemma [HILL99], with a random seed defined in param during the global setup, to extract the entropy from v, then followed by a pseudo-random generator to get a long enough bit-string. Many uses of the same seed in the Leftover-Hash Lemma just lead to a security loss linear in the number of extractions. This gives the following protocol for BlindUSKGen, described in Figure 4.

- The user computes an encryption of the expected identity id and keeps the randomness ρ: $C = \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(\mathsf{id}; \boldsymbol{\rho})\}$.
- For every identity id', the server computes $\mathsf{usk}[\mathsf{id}']$ along with a pair of (secret, public) hash keys $(\mathsf{hk}_{\mathsf{id}'}, \mathsf{hp}_{\mathsf{id}'})$ for a smooth projective hash function on the language $\mathfrak{L}_{C,\mathsf{id}'}$: $\mathsf{hk}_{\mathsf{id}'} = \mathsf{HashKG}(\ell, \mathfrak{L}_{C,\mathsf{id}'}, \mathsf{param})$ and $\mathsf{hp}_{\mathsf{id}'} = \mathsf{ProjKG}(\mathsf{hk}_{\mathsf{id}'}, \ell, (\mathfrak{L}_{C,\mathsf{id}'}, \mathsf{param}))$. He also compute the corresponding hash value $H_{\mathsf{id}'} = \mathsf{Hash}(\mathsf{hk}_{\mathsf{id}'}, (\mathfrak{L}_{C,\mathsf{id}'}, \mathsf{param}), (\ell, C))$. Finally, he sends $(\mathsf{hp}_{\mathsf{id}'}, \mathsf{usk}[\mathsf{id}'] \oplus \mathtt{KDF}(H_{\mathsf{id}'}))$ for every id', where $\oplus$ is a compatible operation.
- Thanks to $\mathsf{hp}_{\mathsf{id}}$, the user is able to compute the corresponding projected hash value $H'_{\mathsf{id}} = \mathsf{ProjHash}(\mathsf{hp}_{\mathsf{id}}, (\mathfrak{L}_{C,\mathsf{id}}, \mathsf{param}), (\ell, C), \boldsymbol{\rho})$. He then recovers $\mathsf{usk}[\mathsf{id}]$ for the initially committed identity id since $H_{\mathsf{id}} = H'_{\mathsf{id}}$.

Fig. 4. Summary of the Generic Construction of $\mathsf{BlindUSKGen}(\langle(\mathcal{S}, \mathsf{msk})(\mathcal{U}, \mathsf{id}, \ell; \boldsymbol{\rho})\rangle)$ for a blind IBE

Theorem 8. *If* IBKEM *is a* PR-ID-CPA*-secure identity-based key encapsulation scheme and* $\mathcal{E}$ *a labeled* CCA*-encryption scheme compatible with an* SPHF*, then* BlindIBKEM *is leak free and weak blind.*

Proof. First, BlindIBKEM satisfies leak-free secret key generation since it relies on the CCA security on the encryption scheme, forbidding a user to open it to another identity than the one initially encrypted. Furthermore, the pseudo-randomness of the SPHF ensures that the blinded user key received for id is indistinguishable from random if he encrypted $\mathsf{id}' \neq \mathsf{id}$. Finally, the weak blindness also relies on the CCA security on the encryption scheme, since an encryption of id is indistinguishable from a encryption of $\mathsf{id}' \neq \mathsf{id}$. □

Sparkling some efficiency. The previous approach allows to achieve our goal, but it has a huge drawback: Since we assume an exponential identity space, it requires an exponential number of answers from the authority. However, if we

focus on the special case of affine IBE with bitwise function[7], a user key can be described as the list $(\mathsf{usk}[0], \mathsf{usk}[0, \mathsf{id}_0], \ldots, \mathsf{usk}[m-1, \mathsf{id}_{m-1}])$ if id_i is the i-th bit of the identity id. One can thus manage to be much more efficient by sending each "bit" evaluation on the user secret key, hidden with a smooth projective hash value on the language "the i-th bit of the identity is a 0 (or 1)", which is common to all identities. We can thus reduce the number of languages from the number of identities (which is exponential) to the length of an identity (which is polynomial). For security reasons, one cannot give directly the evaluation value, but as we are considering the sum of the evaluations for each bit, we simply add a Shamir-like secret sharing, by adding randomness that is going to cancel out at the end.

- The user computes a bit-per-bit encryption of the expected identity id and keeps the randomness ρ: $C = \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(\mathsf{id}; \boldsymbol{\rho})\}$.
- The server computes a fragmented version of all the keys $\mathsf{usk}[\mathsf{id}']$, *i.e.* all the values $\mathsf{usk}[i, b]$ for i from 0 up to the length m of the keys and $b \in \{0, 1\}$. He also computes a pair of (secret, public) hash keys $(\mathsf{hk}_{i,b}, \mathsf{hp}_{i,b})$ for a smooth projective hash function on the language $\mathfrak{L}_{C,i,b}$: "The i-th bit of value encrypted into C is b", *i.e.* $\mathsf{hk}_{i,b} = \mathsf{HashKG}(\ell, \mathfrak{L}_{C,i,b}, \mathsf{param})$ and $\mathsf{hp}_{i,b} = \mathsf{ProjKG}(\mathsf{hk}_{i,b}, \ell, (\mathfrak{L}_{C,i,b}, \mathsf{param}))$. He also computes the corresponding hash value $H_{i,b} = \mathsf{Hash}(\mathsf{hk}_{i,b}, (\mathfrak{L}_{C,i,b}, \mathsf{param}), (\ell, C))$ and chooses random values z_i. Finally, he sends, for each (i, b), $(\mathsf{hp}_{i,b}, \mathsf{busk}[i, b])$, where $\mathsf{busk}[i, b] = \mathsf{usk}[i, b] \oplus \mathtt{KDF}(H_{i,b}) \oplus z_i$, together with $Z = \mathsf{usk}_0 \ominus \left(\bigoplus_i z_i\right)$, where $\oplus$ is a compatible operation and $\ominus$ its inverse.
- Thanks to the $\mathsf{hp}_{i,\mathsf{id}_i}$ for the initially committed identity id, the user is able to compute the corresponding projected hash value $H'_{i,\mathsf{id}_i} = \mathsf{ProjHash}(\mathsf{hp}_{i,\mathsf{id}_i}, (\mathfrak{L}_{C,i,\mathsf{id}_i}, \mathsf{param}), (\ell, C), \boldsymbol{\rho})$, that should be equal to H_{i,id_i} for all i. From the values $\mathsf{busk}[i, \mathsf{id}_i]$, he then recovers $\mathsf{usk}[i, \mathsf{id}_i] \oplus z_i$. Finally, with the operation $\left(\bigoplus_i (\mathsf{usk}[i, \mathsf{id}_i] \oplus z_i)\right) \oplus Z$, he recovers the expected $\mathsf{usk}[\mathsf{id}]$.

Fig. 5. Summary of the Generic Construction of $\mathsf{BlindUSKGen}(\langle(\mathcal{S}, \mathsf{msk})(\mathcal{U}, \mathsf{id}, \ell; \boldsymbol{\rho})\rangle)$ for a Blind affine IBE

Moving on, to be Compatible with the UC Framework. The previous approach may work for a non-UC framework, or if one does not consider adaptive corruptions. However, in this context, interactions should make sense for any possible input chosen by the environment and learnt a posteriori in the simulation during the corruption of an honest party. From the user side, this implies that the last flow should contain enough recoverable information so that a simulator, having sent a commitment to an incorrect identity, can extract the proper user secret key corresponding to the correct identity recovered after the corruption.

[7] They were defined in [BKP14]. Affine IBE derive their name from the fact that only affine operations are done on the identity bits (no hashing, square rooting, inverting... are allowed).

From the server side, this implies that the IBE scheme is defined such as one is able to adapt the user secret keys in order to correspond to the new database learnt a posteriori. Of course, not all schemes allow this property, but this will be the case in the pairing scenario considered in our concrete instanciation.

To deal with corruptions of the user, recall that a simulated server (knowing the secret key of the encryption scheme) is already able to extract the identity committed to. But we now consider that, for all id, $\mathfrak{L}_{\mathsf{id}}$ is the language of the equivocable commitments on words in the inner language $\widetilde{\mathfrak{L}}_{\mathsf{id}} = \{\mathsf{id}\}$. We assume them to be a *Trapdoor Collection of Languages*, which means that it is computationally hard to sample an element in $\mathfrak{L}_1 \cap \cdots \cap \mathfrak{L}_n$, except for the simulator, who possesses a trapdoor tk (the equivocation trapdoor) allowing it to sample an element in the intersection of languages. This allows a simulated user (knowing this trapdoor) not to really bind to any identity during the commitment phase. The only difference with the algorithm described in Figure 5 is that the user now encrypts this word *word* (which is an equivocable commitment on his identity id) rather than directly encrypting his identity id: $C = \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(W; \boldsymbol{\rho})$. This technique is also explained as an application of our OLBE framework, in Appendix F.2 page 49. We will directly prove this protocol during the proof of the oblivious transfer scheme.

3.4 Generic Construction of Adaptive OT

We derive from here our generic construction of OT (depicted in Figure 6). We additionally assume the existence of a Pseudo-Random Generator (PRG) F with input size equal to the plaintext size, and output size equal to the size of the messages in the database and an IND-CPA encryption scheme $\mathcal{E} = (\mathsf{Setup}_{\mathrm{cpa}}, \mathsf{KeyGen}_{\mathrm{cpa}}, \mathsf{Encrypt}_{\mathrm{cpa}}, \mathsf{Decrypt}_{\mathrm{cpa}})$ with plaintext size at least equal to the security parameter. First, the owner of the database generates the keys for such an IBE scheme, and encrypts each line i of the database for the identity i. Then when a user wants to request a given line, he runs the blind user key generation algorithm and recovers the key for the expected given line. This leads to the following security result, proven in Appendix D of the additional content.

Theorem 9. *Assuming that* BlindUSKGen *is constructed as described above, the adaptive Oblivious Transfer protocol described in Figure 6 UC-realizes the functionality $\mathcal{F}^{\mathfrak{L}}_{OT}$ presented in Figure 2 with adaptive corruptions assuming reliable erasures.*

3.5 Pairing-Based Instantiation

Affine Bit-Wise Blind IBE. In [BKP14], the authors propose a generic framework to move from affine Message Authentication Code to IBE, and they propose a tight instantiation of such a MAC, giving an affine bit-wise IBE, which seems like a good candidate for our setting (making it blind and fragmented).

CRS generation:
$\mathsf{crs} \xleftarrow{\$} \mathsf{SetupCom}(1^{\lambda}), \mathsf{param}_{\mathrm{cpa}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cpa}}(1^{\lambda})$.

Database Preparation:

1. Server runs $\mathsf{Gen}(\mathfrak{K})$, to obtain $\mathsf{mpk}, \mathsf{msk}$.
2. For each line t, he computes $(D_t, K_t) = \mathsf{Enc}(\mathsf{mpk}, t)$, and $L_t = K_t \oplus DB(t)$.
3. He also computes $\mathsf{usk}[i, b]$ for all $i = 1 \ldots, m$ and $b = 0, 1$ and erases msk.
4. Server generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathrm{cpa}}(\mathsf{param}_{\mathrm{cpa}})$ for $\mathcal{E}$, stores sk and completely erases the random coins used by KeyGen.
5. He then publishes $\mathsf{mpk}, \{(D_t, L_t)\}_t, \mathsf{pk}$.

Index query on s:

1. User chooses a random value S, computes $R \leftarrow F(S)$ and encrypts S under pk: $c \xleftarrow{\$} \mathsf{Encrypt}_{\mathrm{cpa}}(\mathsf{pk}, S)$
2. User computes $\mathcal{C}$ with the first flow of $\mathsf{BlindUSKGen}(\langle(\mathcal{S}, \mathsf{msk})(\mathcal{U}, s, \ell; \boldsymbol{\rho})\rangle)$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{U}, \mathcal{S})$ (see Figure 5).
3. User stores the random $\boldsymbol{\rho}_s = \{\boldsymbol{\rho}_*\}$ needed to open $\mathcal{C}$ to s, and completely erases the rest, including the random coins used by $\mathsf{Encrypt}_{\mathrm{cpa}}$ and sends $(c, \mathcal{C})$ to the Server

IBE input msk:

1. Server decrypts $S \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and computes $R \leftarrow F(S)$
2. Server runs the second flow of $\mathsf{BlindUSKGen}(\langle(\mathcal{S}, \mathsf{msk})(\mathcal{U}, s, \ell; \boldsymbol{\rho})\rangle)$ on $\mathcal{C}$ (see Figure 5).
3. Server erases every new value except $(\mathsf{hp}_{i,b})_{i,b}, (\mathsf{busk}[i, b])_{i,b}, Z \oplus R$ and sends them over a secure channel.

Data recovery:

1. User then using, $\boldsymbol{\rho}_s$ recovers $\mathsf{usk}[s]$ from the values received from the server.
2. He can then recover the expected information with $\mathsf{Dec}(\mathsf{usk}[s], s, D_s) \oplus L_s$ and erases everything else.

Fig. 6. Adaptive UC-Secure 1-out-of-n OT from a Fragmented Blind IBE

We are thus going to use the family of IBE described in the following picture (Figure 7), which is their instantiation derived from a Naor-Reingold MAC[8]. In the following, $h_i()$ are injective deterministic public functions mapping a bit to a scalar in $\mathbb{Z}_p$.

A property that was not studied in this paper was the blind user key generation: How to generate and answer blind user secret key queries? We answer to this question by proposing the $k-\mathsf{MDDH}$-based variation presented in Figure 8. To fit the global framework we are going to consider the equivocable language of each chameleon hash of the identity bits $(\boldsymbol{a}_i, \boldsymbol{b}_{i,m_i})$, and then a Cramer-Shoup like encryption of $\boldsymbol{b}$ into $\boldsymbol{d}$ (more details in Section B.2). We denote this process as Har in the following protocol, and by $\mathfrak{L}_{\mathsf{Har},i,\mathsf{id}_i}$ the language on identity bits. We thus obtain the following security results.

[8] For the reader familiar with the original result, we combine $x, \boldsymbol{y}$ into a bigger $\boldsymbol{y}$ to lighten the notations, and compact the (x'_i, y'_i) values into a single y' as this has no impact on their construction.

$\underline{\mathsf{Gen}(\mathfrak{K})}$:

$\boldsymbol{A} \xleftarrow{\$} \mathcal{D}_k, \boldsymbol{B} = \overline{\boldsymbol{A}}$
For $i \in [\![0, \ell]\!] : \boldsymbol{Y}_i \xleftarrow{\$} \mathbb{Z}_p^{k+1}; \boldsymbol{Z}_i = \boldsymbol{Y}_i^\top \cdot \boldsymbol{A} \in \mathbb{Z}_p^k$
$\boldsymbol{y}' \xleftarrow{\$} \mathbb{Z}_p^{k+1}; \boldsymbol{z}' = \boldsymbol{y}'^\top \cdot \boldsymbol{A} \in \mathbb{Z}_q^k$
$\mathsf{mpk} := (\mathcal{G}, [\boldsymbol{A}]_1, ([\boldsymbol{Z}_i]_1)_{i \in [\![0,\ell]\!]}, [\boldsymbol{z}']_1)$
$\mathsf{msk} := (\boldsymbol{Y}_i)_{i \in [\![0,\ell]\!]}, \boldsymbol{y}'$
Return $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{USKGen}(\mathsf{msk}, \mathsf{id})}$:

$\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_p^k, \boldsymbol{t} = \boldsymbol{B}\boldsymbol{s}$
$\boldsymbol{w} = (\boldsymbol{Y}_0 \sum_{i=1}^{\ell} \mathsf{id}_i \boldsymbol{Y}_i)\boldsymbol{t} + \boldsymbol{y}' \in \mathbb{Z}_p^{k+1}$
Return $\mathsf{usk}[\mathsf{id}] := ([\boldsymbol{t}]_2, [\boldsymbol{w}]_2) \in \mathbb{G}_2^{k+k+1}$

$\underline{\mathsf{Enc}(\mathsf{mpk}, \mathsf{id})}$:

$\boldsymbol{r} \xleftarrow{\$} \mathbb{Z}_p^k$
$\boldsymbol{c}_0 = \boldsymbol{A}\boldsymbol{r} \in \mathbb{Z}_p^{k+1}$
$\boldsymbol{c}_1 = (\boldsymbol{Z}_0 \sum_{i=0}^{\ell} h_i(\mathsf{id}_i)\boldsymbol{Z}_i) \cdot \boldsymbol{r} \in \mathbb{Z}_p$
$K = \boldsymbol{z}' \cdot \boldsymbol{r} \in \mathbb{Z}_p$.
Return $[K]_T$ and $\mathsf{C} = ([\boldsymbol{c}_0]_1, [\boldsymbol{c}_1]_1) \in \mathbb{G}_1^{k+1+1}$

$\underline{\mathsf{Dec}(\mathsf{usk}[\mathsf{id}], \mathsf{id}, \mathsf{C})}$:

Parse $\mathsf{usk}[\mathsf{id}] = ([\boldsymbol{t}]_2, [\boldsymbol{w}]_2)$
Parse $\mathsf{C} = ([\boldsymbol{c}_0]_1, [\boldsymbol{c}_1]_1)$
$K = e([\boldsymbol{c}_0]_1, [\boldsymbol{w}]_2) \cdot e([\boldsymbol{c}_1]_1, [\boldsymbol{t}]_2)^{-1}$
Return $K \in \mathbb{G}_T$

Fig. 7. A fragmentable affine IBKEM.

– First flow: $\underline{\mathcal{U} \text{ starts by computing}}$
  $\boldsymbol{\rho} \xleftarrow{\$} \mathbb{Z}_p^{1+4\times\ell}$,
  $\boldsymbol{a}, \boldsymbol{d} = \mathsf{Har}(\mathsf{id}, \ell; \rho) \in \mathbb{Z}_p^\ell \times \mathbb{Z}_p^{2\times(k+3)\ell}$,
  Sends $\mathcal{C} = ([\boldsymbol{a}]_1, [\boldsymbol{d}]_2)$ to S
– Second Flow: $\underline{\mathcal{S} \text{ then proceeds}}$
  $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_p^k, \boldsymbol{t} = \boldsymbol{B}\boldsymbol{s}$, $\boldsymbol{f} \xleftarrow{\$} \mathbb{Z}_p^{\ell\times k+1}$,
  For each $i \in [\![1, \lceil \log n \rceil]\!], b \in [\![0, 1]\!]$:
    $\mathsf{hk}_{i,b} = \mathsf{HashKG}(\mathfrak{L}_{\mathsf{Har},i,b}, \boldsymbol{C})$
    $\mathsf{hp}_{i,b} = \mathsf{ProjKG}(\mathsf{hk}_{i,b}, \mathfrak{L}_{\mathsf{Har},i,b}, \boldsymbol{C})$
    $H_{i,b} = \mathsf{Hash}(\mathsf{hk}_{i,b}, \mathfrak{L}_{\mathsf{Har},i,b}, \boldsymbol{C})$
    $\boldsymbol{\omega}_{i,b} = (b\boldsymbol{Y}_i)\boldsymbol{t} + \boldsymbol{f}_i + H_{i,b}$
  Then sets $\boldsymbol{w}_0 = \boldsymbol{Y}_0\boldsymbol{t} + \boldsymbol{y}' - \sum_{i=1}^{\ell} \boldsymbol{f}_i \in \mathbb{Z}_p^{k+1}$
  Returns $\mathsf{busk} := ([\boldsymbol{t}]_2, [\boldsymbol{w}_0]_2, \{[\boldsymbol{\omega}_{i,b}]_2\}, \{[\mathsf{hp}_{i,b}]_2\})$
– $\mathsf{BlindUSKGen}_3$:
  $\underline{\mathcal{U} \text{ then recovers his key}}$
  For each $i \in [\![1, \ell]\!]$:
    $H'_i = \mathsf{ProjHash}(\mathsf{hp}_{i,\mathsf{id}_i}, \mathfrak{L}_{\mathsf{Har},i,\mathsf{id}_i}, \boldsymbol{C}, \rho_i)$
    $\boldsymbol{w}_i = \omega_{i,\mathsf{id}_i} - H'_i$
  $\boldsymbol{w} = \boldsymbol{w}_0 + \sum_{i=1}^{\ell} \boldsymbol{w}_i$
  And then recovers $\mathsf{usk}[\mathsf{id}] := [\boldsymbol{t}]_2, [\boldsymbol{w}]_2$

Fig. 8. $\mathsf{BlindUSKGen}(\langle(\mathcal{S}, \mathsf{msk})(\mathcal{U}, \mathsf{id}, \ell; \boldsymbol{\rho})\rangle)$.

Theorem 10. *This construction achieves both the weak Blindness, and the leak-free secret key generation requirements under the* $k - \mathsf{MDDH}$ *assumption.*

The first one is true under the indistinguishability of the generalized Cramer-Shoup encryption recalled in section B.2 page 38, as the server learns nothing about the line requested during the first flow. It should even be noted that because of the inner chameleon hash, a simulator is able to use the trapdoor to do a commitment to every possible words of the set of languages at once, and so can adaptively decide which id he requested. The proof of the second result is delayed to Appendix C page 39.

For sake of generality, any bit-wise affine IBE could work (like for example Waters IBE [Wat05]), the additional price paid for tightness here is very small and allows to have a better reduction in the proof, but it it not required by the framework itself.

Adaptive UC-Secure Oblivious Transfer. We finally get our instantiation by combining this $k - \mathsf{MDDH}$-based blind IBE with a $k - \mathsf{MDDH}$ variant of El Gamal for the CPA encryption needed. The requirement on the IBE blind user secret key generation (being able to adapt the key if the line changes) is achieved assuming that the server knows the discrete logarithms of the database lines. This is quite easy to achieve by assuming that for all line s, $DB(s) = [db(s)]_1$ where $db(s)$ is the real line (thus known). It implies a few more computation on the user's side in order to recover $db(s)$ from $DB(s)$, but this remains completely feasible if the lines belong to a small space. For practical applications, one could imagine to split all 256-bit lines into 8 pieces for a decent/constant trade-off in favor of computational efficiency.

For $k = 1$, so under the classical SXDH assumption, the first flow requires $8 \log |DB|$ elements in $\mathbb{G}_1$ for the CCA encryption part and $\log(|DB| + 1)$ in $\mathbb{G}_2$ for the chameleon one, while the second flow would now require $1 + 4 \log |DB|$ elements in $\mathbb{G}_1$, $1 + 2 \log |DB|$ for the fragmented masked key, and $2 \log |DB|$ for the projection keys.

4 Oblivious Language-Based Envelope

The previous construction opens new efficient applications to the already known Oblivious-Transfer protocols. But what happens when someone wants some additional access control by requesting extra properties, like if the user is only allowed to ask two lines with the same parity bits, the user can only request lines for whose number has been signed by an authority, or even finer control provided through credentials?

In this section we propose to develop a new primitive, that we call Oblivious Language-Based Envelope (OLBE). The idea generalizes that of Oblivious Transfer and OSBE, recalled right afterwards, for n messages (with n polynomial in the security parameter $\mathfrak{K}$) to provide the best of both worlds.

4.1 Oblivious Signature-Based Envelope

We recall the definition and security requirements of an OSBE protocol given in [LDB03, BPV12], in which a sender $\mathcal{S}$ wants to send a private message $P \in \{0,1\}^{\mathfrak{K}}$ to a recipient $\mathcal{R}$ in possession of a valid certificate/signature on a public message M (given by a certification authority).

Definition 11 (Oblivious Signature-Based Envelope). *An* OSBE *scheme is defined by four algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*, and one interactive protocol* $\mathsf{Protocol}\langle\mathcal{S}, \mathcal{R}\rangle$*:*

- $\mathsf{Setup}(1^{\mathfrak{K}})$*, where* $\mathfrak{K}$ *is the security parameter, generates the global parameters* param*;*
- $\mathsf{KeyGen}(\mathfrak{K})$ *generates the keys* $(\mathsf{vk}, \mathsf{sk})$ *of the certification authority;*
- $\mathsf{Sign}(\mathsf{sk}, M)$ *produces a signature* σ *on the input message* M*, under the signing key* sk*;*
- $\mathsf{Verify}(\mathsf{vk}, M, \sigma)$ *checks whether* σ *is a valid signature on* M*, w.r.t. the public key* vk*; it outputs 1 if the signature is valid, and 0 otherwise.*
- $\mathsf{Protocol}\langle\mathcal{S}(\mathsf{vk}, M, P), \mathcal{R}(\mathsf{vk}, M, \sigma)\rangle$ *between the sender* $\mathcal{S}$ *with the private message* P*, and the recipient* $\mathcal{R}$ *with a certificate* σ*. If* σ *is a valid signature under* vk *on the common message* M*, then* $\mathcal{R}$ *receives* P*, otherwise it receives nothing. In any case,* $\mathcal{S}$ *does not learn anything.*

The authors of [BPV12] proposed some variations to the original definitions from [LDB03], in order to prevent some interference by the authority. Following them, an OSBE scheme should fulfill the following security properties. The formal security games are given in [BPV12]. No UC functionality has already been given, to the best of our knowledge.

- *correct*: the protocol actually allows $\mathcal{R}$ to learn P, whenever σ is a valid signature on M under vk;
- *semantically secure*: the recipient learns nothing about $\mathcal{S}$'s input P if it does not use a valid signature σ on M under vk as input. More precisely, if $\mathcal{S}_0$ owns P_0 and $\mathcal{S}_1$ owns P_1, the recipient that does not use a valid signature cannot distinguish an interaction with $\mathcal{S}_0$ from an interaction with $\mathcal{S}_1$ even if he has eavesdropped on several interactions $\langle\mathcal{S}(\mathsf{vk}, M, P), \mathcal{R}(\mathsf{vk}, M, \sigma)\rangle$ with valid signatures, and the same sender's input P;
- *escrow-free* (*oblivious with respect to the authority*): the authority (owner of the signing key sk), playing as the sender or just eavesdropping, is unable to distinguish whether $\mathcal{R}$ used a valid signature σ on M under vk as input.
- *semantically secure w.r.t. the authority*: after the interaction, the authority (owner of the signing key sk) learns nothing about P from a passive access to a challenge transcript.

4.2 Definition

In such a protocol, a sender $\mathcal{S}$ wants to send one or several private messages (up to $n_{\max} \leq n$) among $(P_1, \ldots, P_n) \in (\{0,1\}^{\ell})^n$ to a recipient $\mathcal{R}$ in possession of a word $W = (W_{i_1}, \ldots, W_{i_{n_{\max}}})$ such that some of the words W_{i_j} may belong to the

corresponding language $\mathfrak{L}_{i_j}$. More precisely, the receiver gets each P_{i_j} as soon as $W_{i_j} \in \mathfrak{L}_{i_j}$ with the requirement that he gets at most $n_{\max}$ messages. In such a scheme, the languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ are assumed to be a trapdoor collection of languages, publicly verifiable and self-randomizable (see Section 2.3 for the definitions of the properties of the languages).

The collections of words can be a single certificate/signature on a message M (encompassing OSBE, with $n = n_{\max} = 1$), a password, a credential, a line number (encompassing 1-out-of-n oblivious transfer[9], with $n_{\max} = 1$), k line numbers (encompassing k-out-of-n oblivious transfer, with $n_{\max} = k$), etc. (see Section F for detailed examples). Following the definitions for OSBE recalled above and given in [LDB03, BPV12], we give the following definition for OLBE. As we consider simulation-based security (in the UC framework), we allow a simulated setup SetupT to be run instead of the classical setup Setup in order to allow the simulator to possess some trapdoors. Those two setup algorithms should be indistinguishable.

Definition 12 (Oblivious Language-Based Envelope). *An* OLBE *scheme is defined by four algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Samp}, \mathsf{Verify})$*, and one interactive protocol* $\mathsf{Protocol}\langle \mathcal{S}, \mathcal{R} \rangle$*:*

- $\mathsf{Setup}(1^{\mathfrak{K}})$*, where* $\mathfrak{K}$ *is the security parameter, generates the global parameters* param*, among which the numbers* n *and* $n_{\max}$*;*

or $\mathsf{SetupT}(1^{\mathfrak{K}})$*, where* $\mathfrak{K}$ *is the security parameter, additionally allows the existence*[10] *of a trapdoor* tk *for the collection of languages* $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$*.*

- $\mathsf{KeyGen}(\mathsf{param}, \mathfrak{K})$ *generates, for all* $i \in \{1, \ldots, n\}$*, the description of the language* $\mathfrak{L}_i$ *(as well as the language key* $\mathsf{sk}_{\mathfrak{L}_i}$ *if need be). If the parameters* param *were defined by* SetupT*, this implicitly also defines the common trapdoor* tk *for the collection of languages* $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$*.*
- $\mathsf{Samp}(\mathsf{param}, I)$ *or* $\mathsf{Samp}(\mathsf{param}, I, (\mathsf{sk}_{\mathfrak{L}_i})_{i \in I})$ *such that* $I \subset \{1, \ldots, n\}$ *and* $|I| = n_{\max}$*, generates a list of words* $(W_i)_{i \in I}$ *such that* $W_i \in \mathfrak{L}_i$ *for all* $i \in I$*;*
- $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ *checks whether* W_i *is a valid word in the language* $\mathfrak{L}_i$*. It outputs* 1 *if the word is valid,* 0 *otherwise;*
- $\mathsf{Protocol}\langle \mathcal{S}((\mathfrak{L}_1, \ldots, \mathfrak{L}_n), (P_1, \ldots, P_n)), \mathcal{R}((\mathfrak{L}_1, \ldots, \mathfrak{L}_n), (W_i)_{i \in I}) \rangle$ *between the sender* $\mathcal{S}$ *with the private messages* $(P_1, \ldots, P_n)$ *and corresponding languages* $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$*, and the recipient* $\mathcal{R}$ *with the same languages and the words* $(W_i)_{i \in I}$ *with* $I \subset \{1, \ldots, n\}$ *and* $|I| = n_{\max}$*, proceeds as follows. For all* $i \in I$*, if the algorithm* $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ *returns* 1*, then* $\mathcal{R}$ *receives* P_i*, otherwise it does not. In any case,* $\mathcal{S}$ *does not learn anything.*

[9] Even if, as explained in the former section, we would rather consider equivocable commitments of line numbers than directly line numbers, in order to get adaptive UC security.

[10] The specific trapdoor will depend on the languages and be computed in the KeyGen algorithm.

4.3 Security Properties and Ideal Functionality

Since we aim at proving the security in the universal composability framework, we now describe the corresponding ideal functionality (depicted in Figure 9). However, in order to ease the comparison with an OSBE scheme, we first list the security properties required, following [LDB03] and [BPV12]:

- *correct*: the protocol actually allows $\mathcal{R}$ to learn $(P_i)_{i\in I}$, whenever $(W_i)_{i\in I}$ are valid words of the languages $(\mathfrak{L}_i)_{i\in I}$, where $I \subset \{1,\ldots,n\}$ and $|I| = n_{\max}$;
- *semantically secure* (*sem*): the recipient learns nothing about the input P_i of $\mathcal{S}$ if it does not use a word in $\mathfrak{L}_i$. More precisely, if $\mathcal{S}_0$ owns $P_{i,0}$ and $\mathcal{S}_1$ owns $P_{i,1}$, the recipient that does not use a word in $\mathfrak{L}_i$ cannot distinguish between an interaction with $\mathcal{S}_0$ and an interaction with $\mathcal{S}_1$ even if the receiver has seen several interactions $\langle \mathcal{S}((\mathfrak{L}_1,\ldots,\mathfrak{L}_n),(P_1,\ldots,P_n)), \mathcal{R}((\mathfrak{L}_1,\ldots,\mathfrak{L}_n),(W'_j)_{j\in I})\rangle$ with valid words $W'_i \in \mathfrak{L}_i$, and the same sender's input P_i;
- *escrow free* (*oblivious with respect to the authority*): the authority corresponding to the language $\mathfrak{L}_i$ (owner of the language secret key $\mathsf{sk}_{\mathfrak{L}_i}$ – if it exists), playing as the sender or just eavesdropping, is unable to distinguish whether $\mathcal{R}$ used a word W_i in the language $\mathfrak{L}_i$ or not. This requirement also holds for anyone holding the trapdoor key tk.
- *semantically secure w.r.t. the authority* (*sem**): after the interaction, the trusted authority (owner of the language secret keys if they exist) learns nothing about the values $(P_i)_{i\in I}$ from the transcript of the execution. This requirement also holds for anyone holding the trapdoor key tk.

Moreover, the Setups should be indistinguishable and it should be infeasible to find a word belonging to two or more languages without the knowledge of tk.

The functionality $\mathcal{F}_{\text{OLBE}}$ is parametrized by a security parameter $\mathfrak{K}$ and a set of languages $(\mathfrak{L}_1,\ldots,\mathfrak{L}_n)$ along with the corresponding public verification algorithms $(\mathsf{Verify}_1,\ldots,\mathsf{Verify}_n)$. It interacts with an adversary $\mathscr{S}$ and a set of parties $\mathfrak{P}_1,\ldots,\mathfrak{P}_N$ via the following queries:

- **Upon receiving an input $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1,\ldots,P_n))$ from party $\mathfrak{P}_i$**, with $P_i \in \{0,1\}^{\mathfrak{K}}$ for all i: record the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1,\ldots,P_n))$ and reveal $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$. Ignore further $\mathtt{Send}$-message with the same ssid from $\mathfrak{P}_i$.
- **Upon receiving an input $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (W_i)_{i\in I})$ where $I \subset \{1,\ldots,n\}$ and $|I| = n_{\max}$ from party $\mathfrak{P}_j$**: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1,\ldots,P_n))$ is not recorded. Otherwise, reveal $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$ and send $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P'_i)_{i\in I})$ to $\mathfrak{P}_j$ where $P'_i = P_i$ if $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ returns 1, and $P'_i = \perp$ otherwise. Ignore further $\mathtt{Received}$-message with the same ssid from $\mathfrak{P}_j$.

Fig. 9. Ideal Functionality for Oblivious Language-Based Envelope $\mathcal{F}_{\text{OLBE}}$

The ideal functionality is parametrized by a set of languages $(\mathfrak{L}_1,\ldots,\mathfrak{L}_n)$. Since we show in the following sections that one can see OSBE and OT as spe-

cial cases of OLBE, it is inspired from the oblivious transfer functionality given in [Can01, CKWZ13, ABB+13] in order to provide a framework consistent with works well-known in the literature. As for oblivious transfer (Figure 2), we adapt them to the simple UC framework for simplicity (this enables us to get rid of `Sent` and `Received` queries from the adversary since the delayed outputs are automatically considered in this simpler framework: We implicitly let the adversary determine if it wants to acknowledge the fact that a message was indeed sent). The first step for the sender (`Send` query) consists in telling the functionality he is willing to take part in the protocol, giving as input his intended receiver and the messages he is willing to send (up to $n_{\max}$ messages). For the receiver, the first step (`Receive` query) consists in giving the functionality the name of the player he intends to receive the messages from, as well as his words. If the word does belong to the language, the receiver recovers the sent message, otherwise, he only gets a special symbol $\perp$.

4.4 Generic UC-Secure Instantiation of OLBE with Adaptive Security

For the sake of clarity, we now concentrate on the specific case where $n_{\max} = 1$. This is the most classical case in practice, and suffices for both OSBE and 1-out-of-n OT. In order to get a generic protocol in which $n_{\max} > 1$, one simply has to run $n_{\max}$ protocols in parallel. This modifies the algorithms Samp and Verify as follows: $\mathsf{Samp}(\mathsf{param}, \{i\})$ or $\mathsf{Samp}(\mathsf{param}, \{i\}, \{\mathsf{sk}_{\mathfrak{L}_i}\})$ generates a word $W = W_i \in \mathfrak{L}_i$ and $\mathsf{Verify}_j(W, \mathfrak{L}_j)$ checks whether W is a valid word in $\mathfrak{L}_j$.

Let us introduce our protocol OLBE: we will call $\mathcal{R}$ the receiver and $\mathcal{S}$ the sender. If $\mathcal{R}$ is an honest receiver, then he knows a word $W = W_i$ in one of the languages $\mathfrak{L}_i$. If $\mathcal{S}$ is an honest sender, then he wants to send up a message among $(P_1, \ldots, P_n) \in (\{0,1\}^{\mathfrak{K}})^n$ to $\mathcal{R}$. We assume the languages $\mathfrak{L}_i$ to be self-randomizable and publicly verifiable. We also assume the collection of languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ possess a trapdoor, that the simulator is able to find by programming the common reference string. As recalled in the previous section, this trapdoor enables him to find a word lying in the intersection of the n languages. This should be infeasible without the knowledge of the trapdoor. Intuitively, this allows the simulator to commit to all languages at once, postponing the time when it needs to choose the exact language he wants to bind to. On the opposite, if a user was granted the same possibilities, this would prevent the simulator to extract the chosen language.

We assume the existence of a labeled CCA-encryption scheme $\mathcal{E} = (\mathsf{Setup}_{\mathrm{cca}}, \mathsf{KeyGen}_{\mathrm{cca}}, \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}, \mathsf{Decrypt}^{\ell}_{\mathrm{cca}})$ compatible with an SPHF onto a set G. In the KeyGen algorithm, the description of the languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ thus implicitly defines the languages $(\mathfrak{L}_{C,1}, \ldots, \mathfrak{L}_{C,n})$ of CCA-encryptions of elements of $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$. We additionally use a key derivation function `KDF` to derive a pseudo-random bit-string $K \in \{0,1\}^{\mathfrak{K}}$ from a pseudo-random element $v \in G$. One can use the Leftover-Hash Lemma [HILL99], with a random seed defined in param during the global setup, to extract the entropy from v, then followed by a pseudo-random generator to get a long enough bit-string. Many uses of

the same seed in the Leftover-Hash Lemma just lead to a security loss linear in the number of extractions. We also assume the existence of a Pseudo-Random Generator (PRG) F with input size equal to the plaintext size, and output size equal to the size of the messages in the database and an IND-CPA encryption scheme $\mathcal{E} = (\mathsf{Setup}_{\mathrm{cpa}}, \mathsf{KeyGen}_{\mathrm{cpa}}, \mathsf{Encrypt}_{\mathrm{cpa}}, \mathsf{Decrypt}_{\mathrm{cpa}})$ with plaintext size at least equal to the security parameter.

We follow the ideas of the oblivious transfer constructions given in [ABB+13, BC15], giving the protocol presented on Figure 10. For the sake of simplicity, we only give the version for adaptive security, in which the sender generates a public key pk and ciphertext c to create a somewhat secure channel (they would not be used in the static version).

CRS: $\mathsf{param} \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}})$, $\mathsf{param}_{\mathrm{cca}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cca}}(1^{\mathfrak{K}})$, $\mathsf{param}_{\mathrm{cpa}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cpa}}(1^{\mathfrak{K}})$.

Pre-flow:

1. Sender generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathrm{cpa}}(\mathsf{param}_{\mathrm{cpa}})$ for $\mathcal{E}$, stores sk and completely erases the random coins used by KeyGen.
2. Sender sends pk to User.

Flow From the Receiver $\mathcal{R}$:

1. User chooses a random value J, computes $R \leftarrow F(J)$ and encrypts J under pk: $c \xleftarrow{\$} \mathsf{Encrypt}_{\mathrm{cpa}}(\mathsf{pk}, J)$.
2. User computes $C \xleftarrow{\$} \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(W; r)$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{R}, \mathcal{S})$.
3. User completely erases J and the random coins used by $\mathsf{Encrypt}_{\mathrm{cpa}}$ and sends C and c to Sender. He also checks the validity of his words: the receiver only keeps the random coins used by $\mathsf{Encrypt}_{\mathrm{cca}}$ for the j such that $\mathsf{Verify}_j(W, \mathfrak{L}_j) = 1$ (since he knows they will be useless otherwise).

Flow From the Sender $\mathcal{S}$:

1. Sender decrypts $J \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and then $R \leftarrow F(J)$.
2. For all $j \in \{1, \ldots, n\}$, sender computes $\mathsf{hk}_j = \mathsf{HashKG}(\ell, \mathfrak{L}_{C,j}, \mathsf{param})$, $\mathsf{hp}_j = \mathsf{ProjKG}(\mathsf{hk}_j, \ell, (\mathfrak{L}_{C,j}, \mathsf{param}))$, $v_j = \mathsf{Hash}(\mathsf{hk}_j, (\mathfrak{L}_{C,j}, \mathsf{param}), (\ell, C))$, $Q_j = P_j \oplus \mathtt{KDF}(v_j) \oplus R$.
3. Sender erases everything except $(Q_j, \mathsf{hp}_j)_{j \in \{1,\ldots,n\}}$ and sends them over a secure channel.

Message recovery:

Upon receiving $(Q_j, \mathsf{hp}_j)_{j \in \{1,\ldots,n\}}$, $\mathcal{R}$ can recover P_i by computing $P_i = Q_i \oplus \mathsf{ProjHash}(\mathsf{hp}_i, (\mathfrak{L}_{C,i}, \mathsf{param}), (\ell, C), r) \oplus R$.

Fig. 10. UC-Secure OLBE for One Message (Secure Against Adaptive Corruptions)

Theorem 13. *The oblivious language-based envelope scheme described in Figure 10 is* UC*-secure in the presence of adaptive adversaries, assuming reliable erasures, an* IND-CPA *encryption scheme, and an* IND-CCA *encryption scheme admitting an* SPHF *on the language of valid ciphertexts of elements of $\mathfrak{L}_i$ for all i, as soon as the languages are self-randomizable, publicly-verifiable and admit a common trapdoor. The proof is given in Appendix E of the additional content.*

4.5 Oblivious Primitives Obtained by the Framework

Classical oblivious primitives such as Oblivious Transfer (both 1-out-of-n and k-out-of-n) or Oblivious Signature-Based Envelope directly lie in this framework and can be seen as examples of Oblivious Language-Based Envelope. We provide in Appendix F of the additional content details about how to describe the languages and choose appropriate smooth projective hash functions to readily achieve current instantiations of Oblivious Signature-Based Envelope or Oblivious Transfer from our generic protocol. The framework also enables us to give a new instantiation of Access Controlled Oblivious Transfer under classical assumptions. In such a primitive, the user does not automatically gets the line he asks for, but has to prove that he possesses one of the credential needed to access this particular line.

For the sake of simplicity, all the instantiations given are pairing-based but techniques explained in [BC15] could be used to rely on other families of assumptions, like decisional quadratic residue or even LWE.

References

[ABB+13] Michel Abdalla, Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, and David Pointcheval. SPHF-friendly non-interactive commitments. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 214–234. Springer, December 2013.

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, August 2009.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135. Springer, May 2001.

[AP06] Michel Abdalla and David Pointcheval. A scalable password-based group key exchange protocol in the standard model. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 332–347. Springer, December 2006.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, May 2014.

[BBC+13a] Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient UC-secure authenticated key-exchange for algebraic languages. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 272–291. Springer, February / March 2013.

[BBC+13b] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, August 2013.

[BC15] Olivier Blazy and Céline Chevalier. Generic construction of uc-secure oblivious transfer. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 65–86. Springer, 2015.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, August 2001.

[BFPV10] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Rosario Gennaro, editor, *Proceedings of PKC 2011*, Lecture Notes in Computer Science. Springer, 2010. Full version available from the web page of the authors.

[BG13] Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 646–663. Springer, May 2013.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, February 2005.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[BKP14] Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 408–425. Springer, August 2014.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*. Springer, 1998. Invited paper.

[BPV12] Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 94–111. Springer, March 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCGS10] Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. Credential authenticated identification and key exchange. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 255–276. Springer, August 2010.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In *CRYPTO 2015, Part II*, LNCS, pages 3–22. Springer, August 2015.

[CDH12] Jan Camenisch, Maria Dubovitskaya, and Kristiyan Haralambiev. Efficient structure-preserving signature scheme from standard assumptions. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 76–94. Springer, September 2012.

[CDN09] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Oblivious transfer with access control. In Ehab Al-Shaer, Somesh Jha, and Angelos D.

Keromytis, editors, *ACM CCS 09*, pages 131–140. ACM Press, November 2009.

[CDNZ11] Jan Camenisch, Maria Dubovitskaya, Gregory Neven, and Gregory M. Zaverucha. Oblivious transfer with hidden access control policies. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 192–209. Springer, March 2011.

[CFH+07] Yang Cui, Eiichiro Fujisaki, Goichiro Hanaoka, Hideki Imai, and Rui Zhang. Formal security treatments for signatures from identity-based encryption. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 218–227. Springer, November 2007.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995.

[CKP07] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 613–630. Springer, August 2007.

[CKWZ13] Seung Geol Choi, Jonathan Katz, Hoeteck Wee, and Hong-Sheng Zhou. Efficient, adaptively secure, and composable oblivious transfer with a single, global CRS. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 73–88. Springer, February / March 2013.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, August 1998.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, April / May 2002.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DOR99] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 74–89. Springer, May 1999.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, August 2013.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, August 1984.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, August 2006.

[GD14] Vandana Guleria and Ratna Dutta. Lightweight universally composable adaptive oblivious transfer. In ManHo Au, Barbara Carminati, and C.-C.Jay Kuo, editors, *Network and System Security*, volume 8792 of *Lecture*

Notes in Computer Science, pages 285–298. Springer International Publishing, 2014.

[GH07] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 265–282. Springer, December 2007.

[GH08] Matthew Green and Susan Hohenberger. Universally composable adaptive oblivious transfer. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 179–197. Springer, December 2008.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *30th ACM STOC*, pages 151–160. ACM Press, May 1998.

[GKW15] Romain Gay, Iordanis Kerenidis, and Hoeteck Wee. Communication complexity of conditional disclosure of secrets and attribute-based encryption. In *CRYPTO 2015, Part II*, LNCS, pages 485–502. Springer, August 2015.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, April 2008.

[Har11] Kristiyan Haralambiev. *Efficient Cryptographic Primitives for Non-Interactive Zero-Knowledge Proofs and Applications*. PhD thesis, New York University, 2011.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, August 2007.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662. Springer, July 2014.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, March 2009.

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 78–95. Springer, May 2005.

[KLL+15] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. *PoPETs*, 2015(2):222–243, 2015.

[KNP11] Kaoru Kurosawa, Ryo Nojima, and Le Trieu Phong. Generic fully simulatable adaptive oblivious transfer. In Javier Lopez and Gene Tsudik, editors, *ACNS 11*, volume 6715 of *LNCS*, pages 274–291. Springer, June 2011.

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, March 2011.

[LDB03] Ninghui Li, Wenliang Du, and Dan Boneh. Oblivious signature-based envelope. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *22nd ACM PODC*, pages 182–189. ACM, July 2003.

[LL07] Sven Laur and Helger Lipmaa. A new protocol for conditional disclosure of secrets and its applications. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 207–225. Springer, June 2007.

[NP97] Moni Naor and Benny Pinkas. Visual authentication and identification. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 322–336. Springer, August 1997.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, August 1992.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, August 2008.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR81, Harvard University, 1981.

[RKP09] Alfredo Rial, Markulf Kohlweiss, and Bart Preneel. Universally composable adaptive priced oblivious transfer. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pages 231–247. Springer, August 2009.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, August 1992.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, August 1984.

[Sha07] Hovav Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074, 2007. `http://eprint.iacr.org/2007/074.pdf`.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, May 2005.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, February 2014.

[WHC+14] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. SCORAM: Oblivious RAM for secure computation. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 191–202. ACM Press, November 2014.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

Additional Content

A Classical Primitives

Digital Signature.

A digital signature scheme $\mathcal{S}$ [DH76, GMR88] allows a signer to produce a verifiable proof that he indeed produced a message. It is de scribed through four algorithms:

Definition 14 (Digital Signature Scheme). $\sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(1^{\mathfrak{K}})$ *where* $\mathfrak{K}$ *is the security parameter, generates the global parameters* param *of the scheme, for example the message space;*
- $\mathsf{KeyGen}(\mathsf{param})$*, outputs a pair of* $(\mathsf{sk}, \mathsf{vk})$*, where* sk *is the (secret) signing key, and* vk *is the (public) verification key;*
- $\mathsf{Sign}(\mathsf{sk}, M; \mu)$*, outputs a signature* $\sigma(M)$*, on a message* M*, under the signing key* sk*, and some randomness* μ*;*
- $\mathsf{Verify}(\mathsf{vk}, M, \sigma)$ *checks the validity of the signature* σ *with respect to the message* M *and the verification key* vk*. And so outputs a bit.*

In the following we will expect at least two properties for signatures:

- *Correctness*: For every pair $(\mathsf{vk}, \mathsf{sk})$ generated by KeyGen, for every message M, and for all randomness μ, we have $\mathsf{Verify}(\mathsf{vk}, M, \mathsf{Sign}(\mathsf{sk}, M; \mu)) = 1$.
- *Existential Unforgeability under Chosen Message Attacks [GMR88]* ($\mathsf{EUF} - \mathsf{CMA}$). Even after querying n valid signatures on chosen messages (M_i), an adversary should not be able to output a valid signature on a fresh message M. To formalize this notion, we define a signing oracle OSign:
  - $\mathsf{OSign}(\mathsf{vk}, m)$: This oracle outputs a signature on m valid under the verification key vk. The requested message is added to the signed messages set $\mathcal{SM}$.

$\mathsf{Exp}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}, \mathsf{OSign}(\mathsf{vk}, \cdot))$
4. $b \leftarrow \mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*)$
5. `IF` $m^* \in \mathcal{SM}$ `RETURN` 0
6. `ELSE RETURN` b

The probability of success against this game is denoted by $\mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = 1]$, $\mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S}}(\mathfrak{K}, t) = \max_{\mathcal{A} \leq t} \mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K})$.

Encryption. An encryption scheme $\mathcal{C}$ is described through four algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key pk and a (private) decryption key dk;
- $\mathsf{Encrypt}(\mathsf{ek}, M; \rho)$ outputs a ciphertext $\mathcal{C}$, on M, under the encryption key pk, with the randomness ρ;
- $\mathsf{Decrypt}(\mathsf{dk}, \mathcal{C})$ outputs the plaintext M, encrypted in the ciphertext $\mathcal{C}$ or $\perp$.

Such encryption scheme is required to have the following security properties:

– *Correctness*: For every pair of keys $(\mathsf{ek}, \mathsf{dk})$ generated by KeyGen, every messages M, and every random ρ, we should have $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, M; \rho)) = M$.
– *Indistinguishability under Adaptive Chosen Ciphertext Attack* IND-CCA ([NY90, RS92]):
  • IND-CCA: An adversary should not be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts, and ask several decryption of ciphertexts different from challenge one.
  The $\mathsf{ODecrypt}$ oracle outputs the decryption of c under the challenge decryption key dk. The input queries (c) are added to the list $\mathcal{CT}$ of decrypted ciphertexts.

$\mathsf{Exp}^{\mathsf{ind\text{-}cca}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(M_0, M_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk}, \mathsf{ODecrypt}(\cdot))$
4. $c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : c^*, \mathsf{ODecrypt}(\cdot))$
6. `IF` $(c^*) \in \mathcal{CT}$ `RETURN` 0
7. `ELSE RETURN` b'

Commitment. Commitments allow a user to commit to a value without revealing it, but without the possibility to later change his mind. It is composed of these algorithms:
– $\mathsf{SetupCom}(1^{\mathfrak{K}})$ generates the system parameters, according to the security parameter $\mathfrak{K}$.
– $\mathsf{KeyGen}(\mathsf{param})$ generates a commitment key ck, and possibly some verification key vk.
– $\mathsf{Commit}(\mathsf{ck}, \mathsf{vk}, m; r)$ produces a commitment c on the input message $m \in \mathcal{M}$ using the random coins $r \xleftarrow{\$} \mathcal{R}$.
– $\mathsf{Decommit}(\mathsf{ck}, c, m; r)$ opens the commitment c and reveals the message m.

Such a commitment scheme should be both *hiding*, which says that the commit phase does not leak any information about m, and *binding*, which says that the decommit phase should not be able to open to two different messages. Additional features are also sometimes required, such as non-malleability, extractability, and/or equivocability. We may also include a label ℓ, which is an additional public information that has to be the same in both the commit and the decommit phases.

A commitment scheme is said *equivocable* if it has a second setup $\mathsf{SetupComT}(1^{\mathfrak{K}})$ that additionally outputs a trapdoor τ, and two algorithms

– $\mathsf{SimCom}^{\ell}(\tau)$ that takes as input the trapdoor τ and a label ℓ and outputs a pair (C, eqk), where C is a commitment and eqk an equivocation key;
– $\mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, x)$ that takes as input a commitment C, a label ℓ, a message x, an equivocation key eqk, and outputs an opening data δ for C and ℓ on x.

such as the following properties are satisfied: *trapdoor correctness* (all simulated commitments can be opened on any message), *setup indistinguishability* (one cannot distinguish the CRS ρ generated by $\mathsf{SetupCom}$ from the one generated

by $\mathsf{SetupComT}$) and *simulation indistinguishability* (one cannot distinguish a real commitment (generated by Com) from a fake commitment (generated by SCom), even with oracle access to fake commitments), denoting by SCom the algorithm that takes as input the trapdoor τ, a label ℓ and a message x and which outputs $(C, \delta) \xleftarrow{\$} \mathsf{SCom}^{\ell}(\tau, x)$, computed as $(C, \mathsf{eqk}) \xleftarrow{\$} \mathsf{SimCom}^{\ell}(\tau)$ and $\delta \leftarrow \mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, x)$.

A commitment scheme $\mathcal{C}$ is said *extractable* if it has a second setup $\mathsf{SetupComT}(1^{\mathfrak{K}})$ that additionally outputs a trapdoor τ, and a new algorithm

- $\mathsf{ExtCom}^{\ell}(\tau, C)$ which takes as input the trapdoor τ, a commitment C, and a label ℓ, and outputs the committed message x, or $\perp$ if the commitment is invalid.

such as the following properties are satisfied: *trapdoor correctness* (all commitments honestly generated can be correctly extracted: for all ℓ, x, if $(C, \delta) \xleftarrow{\$} \mathsf{Com}^{\ell}(x)$ then $\mathsf{ExtCom}^{\ell}(C, \tau) = x$), *setup indistinguishability* (as above) and *binding extractability* (one cannot fool the extractor, *i.e.*, produce a commitment and a valid opening data to an input x while the commitment does not extract to x).

Chameleon Hash. This directly echoes to Chameleon Hashes, traditionally defined by three algorithms $\mathsf{CH} = (\mathsf{KeyGen}, \mathsf{CH}, \mathsf{Coll})$:

- $\mathsf{KeyGen}(\mathfrak{K})$: Outputs the chameleon hash key ck and the trapdoor tk;
- $\mathsf{CH}(\mathsf{ck}, m; r)$: Picks a random r, and outputs the chameleon hash a.
- $\mathsf{Coll}(\mathsf{ck}, m, r, m', \mathsf{tk})$: Takes as input the trapdoor tk, a start message and randomness pair (m, r) and a target message m' and outputs a target randomness r' such that $\mathsf{CH}(\mathsf{ck}, m; r) = \mathsf{CH}(\mathsf{ck}, m'; r')$.

The standard security notion for CH is collision resistance. Formally, CH is $(t, \varepsilon) - \mathsf{coll}$ if for the adversary $\mathcal{A}$ running in time at most t we have:

$$\Pr\left[\begin{array}{c}(\mathsf{ck}, \mathsf{tk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathfrak{K}); ((m_1, r_1), (m_2, r_2)) \xleftarrow{\$} \mathcal{A}(\mathsf{ck}) \\ \wedge \quad \mathsf{CH}(\mathsf{ck}, m_1; r_1) = \mathsf{CH}(\mathsf{ck}, m_2; r_2) \wedge m_1 \neq m_2\end{array}\right] \leq \varepsilon.$$

However, any user in possession of the trapdoor tk is able to find a collision using Coll. Additionally, Chameleon Hash functions have the uniformity property, which means the hash value leaks nothing about the message input. Formally, for all pair of messages m_1 and m_2 and the randomly chosen r, the probability distributions of the random variables $\mathsf{CH}(\mathsf{ck}, m_1, r)$ and $\mathsf{CH}(\mathsf{ck}, m_2, r)$ are computationally indistinguishable.

We need here the hash value to be verifiable, so that we add two $\mathsf{VKeyGen}$ and Valid algorithms (executed by the receiver).

- $\mathsf{VKeyGen}(\mathsf{ck})$: Outputs the chameleon designated verification key vk by appending it to ck and the trapdoor vtk. This trapdoor can be empty or public if the chameleon hash is publicly verifiable.
- $\mathsf{Valid}(\mathsf{ck}, m, a, d, \mathsf{vtk})$: Allows to check that the sender knows how to open a Chameleon Hash a to a specific value m for the witness d. The verification can be public if vtk is empty or public, or specific to the receiver otherwise.

B Building Blocks

B.1 Classical Building Blocks

Waters Signature To sign scalar message in the standard model, one can use Waters Signatures [Wat05]. This signature scheme is defined by four algorithms:

Definition 15 (Waters Signature Scheme). $\mathcal{S} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(1^{\mathfrak{K}})$*, where* $\mathfrak{K}$ *is the security parameter, generates the global parameters* param *of the scheme, and more specifically the bilinear group* $(p, \mathbb{G}, \mathbb{G}_T, e, g)$*, an extra generator* h*, and generators* $(u_i)_{[\![0,k]\!]}$ *for the Waters function, where* k *is a polynomial in* $\mathfrak{K}$*,* $\mathcal{F}(m) = u_0 \prod_{i \in [\![1,k]\!]} u_i^{m_i}$*, where* $m = (m_1, \ldots, m_k) \in \{0,1\}^k$*.*
- $\mathsf{KeyGen}(\mathsf{param})$ *picks a random* $x \xleftarrow{\$} \mathbb{Z}_p$ *and outputs the secret key* $\mathsf{sk} = Y = h^x$*, and the verification key* $\mathsf{vk} = X = g^x$*;*
- $\mathsf{Sign}(\mathsf{sk}, m; \mu)$ *outputs a signature* $\sigma(m) = (Y\mathcal{F}(m)^{\mu}, g^{-\mu})$*;*
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ *checks the validity of* σ*, by checking if the following pairing equation holds:* $e(g, \sigma_1) \cdot e(\mathcal{F}(m), \sigma_2) \stackrel{?}{=} e(X, h)$

Theorem 16. *This scheme is* $\mathsf{EUF} - \mathsf{CMA}$ *under the* CDH *assumption.*

Theorem 17. *Waters Signature is randomizable if we define:*

- $\mathsf{Random}(\mathsf{vk}, \mathcal{F}(m), \sigma = (\sigma_1, \sigma_2); \mu')$ *outputs* $\sigma' = (\sigma_1 \cdot \mathcal{F}(m)^{\mu'}, \sigma_2 \cdot g^{-\mu'})$*.*

Proof. We simply have:

$$\sigma = \mathsf{Sign}(\mathsf{sk}, \mathcal{F}(m); \mu) \quad \Rightarrow \quad \mathsf{Random}(\mathsf{vk}, \mathcal{F}(m), \sigma; \mu') = \mathsf{Sign}(\mathsf{sk}, \mathcal{F}(m); \mu + \mu' \mod p).$$

Due to the additive law in $\mathbb{Z}_p$, fresh signature distribution is indistinguishable from randomized one. □

CDH-based Chameleon Hash [BC15]

- $\mathsf{KeyGen}(\mathfrak{K})$: Outputs the chameleon hash key $\mathsf{ck} = (g, h)$ and the trapdoor $\mathsf{tk} = \alpha$, where $g^{\alpha} = h$;
- $\mathsf{VKeyGen}(\mathsf{ck})$: Appends f to ck and $\mathsf{vtk} = \log_g(f)$
- $\mathsf{CH}(\mathsf{ck}, m; r)$: Picks a random $r \in \mathbb{Z}_p$, and outputs the chameleon hash $a = h^r g^m$. Sets $d = f^r$.
- $\mathsf{Coll}(m, r, m', \mathsf{tk})$: outputs $r' = r + (m - m')/\alpha$.
- $\mathsf{Valid}(\mathsf{ck}, m, a, d, \mathsf{vtk})$: One can check if $a = h^m \cdot d^{1/\mathsf{vtk}}$.

In a pairing environment, there is a trivial way to check this CH using a pairing instead of knowing vtk.

This is a CDH variant of the Pedersen chameleon hash [Ped92].

ElGamal Encryption ElGamal encryption [ElG84] is defined by the following four algorithms:

– $\mathsf{Setup}(1^{\mathfrak{K}})$: The scheme needs a multiplicative group $(p, \mathbb{G}, g)$,. The global parameters param consist of these elements $(p, \mathbb{G}, g)$.
– $\mathsf{KeyGen}(\mathsf{param})$: Chooses one random scalar $\mu \xleftarrow{\$} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = \mu$, and the public key $\mathsf{ek} = X = g^{\mu}$.
– $\mathsf{Encrypt}(\mathsf{ek} = X, M; \alpha)$: For a message $M \in \mathbb{G}$ and a random scalar $\alpha \xleftarrow{\$} \mathbb{Z}_p$, computes the ciphertext as $\boldsymbol{c} = \big(c_1 = X^{\alpha}M, c_2 = g^{\alpha}\big)$.
– $\mathsf{Decrypt}(\mathsf{dk} = \mu, \boldsymbol{c} = (c_1, c_2))$: One computes $M = c_1/(c_2^{\mu})$.

As shown by Boneh [Bon98], this scheme is IND-CPA under the hardness of DDH.

Cramer-Shoup Encryption The Cramer-Shoup encryption scheme [CS98] is an IND-CCA version of the ElGamal Encryption. We present it here as a labeled public-key encryption scheme, the classical version is done with $\ell = \emptyset$.

– $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with a generator g
– $\mathsf{KeyGen}(\mathsf{param})$ generates $(g_1, g_2) \xleftarrow{\$} \mathbb{G}^2$, $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \xleftarrow{\$} \mathbb{Z}_p^5$, and sets, $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^{z}$. It also chooses a Collision-Resistant hash function $\mathfrak{H}_K$ in a hash family $\mathcal{H}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (g_1, g_2, c, d, h, \mathfrak{H}_K)$.
– $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (\ell, \mathbf{u} = (g_1^r, g_2^r), e = M \cdot h^r, v = (cd^{\xi})^r)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.
– $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$: one first computes $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$ and checks whether $u_1^{x_1+\xi y_1} \cdot u_2^{x_2+\xi y_2} \stackrel{?}{=} v$. If the equality holds, one computes $M = e/(u_1^{z})$ and outputs M. Otherwise, one outputs $\perp$.

The security of the scheme is proven under the DDH assumption and the fact the hash function used is a Universal One-Way Hash Function. A generalization of this encryption to the $k-$MDDH assumption can be found further below.

Smooth Projective Hash Functions on Cramer Shoup Encryption
One can now build a Hash Proof system on this CCA-2 scheme:

– $\mathsf{HashKG}(\mathfrak{L}_M) : \mathsf{hk} \xleftarrow{\$} \mathbb{Z}_p^4$,
– $\mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}_M, \ell, [\mathcal{C}]_2)) :$ Setting $\mathsf{ht} = (h, g_1, g_2, cd^{\theta})$, we have $\mathsf{hp} = h^{\mathsf{hk}_1} g_1^{\mathsf{hk}_2} g_2^{\mathsf{hk}_3} \left(cd^{\theta}\right)^{\mathsf{hk}_4}$, where $\theta = H(\ell, \boldsymbol{e})$
– $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}_M, \ell, [\mathcal{C}]_2)) : H \leftarrow (\mathcal{C}_1/M)^{\mathsf{hk}_1} \mathcal{C}_2^{\mathsf{hk}_2} \mathcal{C}_3^{\mathsf{hk}_3} \mathcal{C}_4^{\mathsf{hk}_4}$,
– $\mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}_M, \ell, [\mathcal{C}]_2), r) H' \leftarrow \mathsf{hp}^r$

B.2 k-MDDH Building Blocks

In this section we extend classical building blocks to the MDDH assumptions. While most of them were at least implicitly defined before in [EHK+13], we feel it may be useful to group them together for ease of reading.

Chameleon Hash We first extend the Pedersen commitment, to obtain a compatible verifiable Chameleon Hash functions:

– $\mathsf{KeyGen}(\mathfrak{K})$: Outputs the chameleon hash key $\mathsf{ck}_1 = \boldsymbol{F} \xleftarrow{\$} \mathcal{D}_k$ and the trapdoor $\mathsf{tk} = \underline{\boldsymbol{F}} \cdot \overline{\boldsymbol{F}^{-1}}$, it also generates $\boldsymbol{G} \xleftarrow{\$} \mathsf{GL}_k$ and adds $\mathsf{ck}_2 = [\boldsymbol{EG}]_2$ to the key ck and keeps the verification trap $\mathsf{vtk} = \boldsymbol{G}^{-1}$
– $\mathsf{CH}(\mathsf{ck}, m; \boldsymbol{\rho})$: Picks a random $\boldsymbol{\rho} \in \mathbb{Z}_p^{k+1}$, and outputs the chameleon hash $[\boldsymbol{a}]_2 = [(\boldsymbol{\rho}^\top \mid m)\mathsf{ck}_1]_2$. Sets $\boldsymbol{d} = \boldsymbol{\rho}^\top \mathsf{ck}_2$.
– $\mathsf{Coll}(m, \boldsymbol{\rho}, m', \mathsf{tk})$: outputs $\boldsymbol{\rho'} = \boldsymbol{\rho} + (m - m')\mathsf{tk}$.
– $\mathsf{Valid}(\mathsf{ck}, m, [\boldsymbol{a}]_2, [\boldsymbol{d}]_2, \mathsf{vtk})$: The user outputs $[\boldsymbol{d}]_2$, so that one can check if $\boldsymbol{a} = \boldsymbol{d}^\top \mathsf{vtk} + \boldsymbol{m}\underline{\boldsymbol{F}}$.

Correctness follows easily, finding a collision leads to directly to computing tk, and so $\boldsymbol{G}$ from $[\boldsymbol{E}]_2$ and vtk. Such Pedersen Commitment was already used in the master public key generation in [BKP14]. It also corresponds to the $k - \mathsf{MDDH}$ version of the Haralambiev [Har11, Section 4.1.4] TC4 commitment scheme, called TC4 which was revisited in recent works [ABB+13, BC15] as the basis for a UC secure commitment.

k-MDDH Linear Encryption [EHK+13] provides a CPA encryption scheme:

– $\mathsf{KeyGen}(\mathfrak{K})$: Picks $\boldsymbol{E} \xleftarrow{\$} \mathcal{D}_k$, sets $\mathsf{pk} = [\boldsymbol{E}]_2, \mathsf{sk} = (\underline{\boldsymbol{E}} \cdot \overline{\boldsymbol{E}^{-1}})$ to be the public encryption key.
– $\mathsf{Encrypt}(\mathsf{ek}, M, \ell; \boldsymbol{\mu})$ If $M \in \mathbb{G}_2$, $\boldsymbol{e} = \mathsf{pk}\boldsymbol{\mu} + \begin{bmatrix} 0 \\ M \end{bmatrix}_2$.
– $\mathsf{Decrypt}(\mathsf{dk}, \boldsymbol{e}, \ell)$ Outputs $M = \mathsf{sk}\overline{\boldsymbol{e}} + \underline{\boldsymbol{e}}$.

$2m$ labelled k-MDDH Multi Cramer Shoup Encryption We are going to need a CCA-2 encryption, SPHF friendly, and proven under MDDH. Fortunately for us, [EHK+13] also provides a compatible Universal$_2$ Hash Proof system, which thanks to [CS02] leads to the required scheme:

– $\mathsf{KeyGen}(\mathfrak{K})$: Picks $\boldsymbol{E} \xleftarrow{\$} \mathcal{D}_k, \boldsymbol{u}, \boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_p^{k+1}$, sets $\mathsf{pk} = ([\boldsymbol{E}]_2, [\boldsymbol{u}^\top \boldsymbol{E}]_2, [\boldsymbol{v}^\top \boldsymbol{E}]_2), \mathsf{sk} = (\underline{\boldsymbol{E}} \cdot \overline{\boldsymbol{E}^{-1}}, \boldsymbol{u}, \boldsymbol{v})$ to be the public encryption key, where $\mathcal{H}$ is a random collision-resistant hash function from $\mathcal{H}$ [11].
– $\mathsf{Encrypt}(\mathsf{ek}, M, \ell; \boldsymbol{\mu})$ If $M \in \mathbb{G}_2$, $\mathcal{C} = (\boldsymbol{e} = \mathsf{pk}_1\boldsymbol{\mu} + \begin{bmatrix} 0 \\ M \end{bmatrix}_2, w = [(\mathsf{pk}_2 + \theta\mathsf{pk}_3)\boldsymbol{\mu}]_2$. where $\theta = H(\ell, \boldsymbol{e})$.
– $\mathsf{Decrypt}(\mathsf{dk}, \mathcal{C}, \ell)$ If $[w]_2 \stackrel{?}{=} [(\boldsymbol{u}^\top + \theta \boldsymbol{v}^\top)\overline{\boldsymbol{e}}]_2$, then outputs $M = \mathsf{sk}\overline{\boldsymbol{e}} + \underline{\boldsymbol{e}}$.

The above scheme can be extended naturally to encrypt matrices of group elements $\boldsymbol{D} = (\boldsymbol{D}_1, \ldots, \boldsymbol{D}_{2m}) \in \mathbb{G}_2^{2m}$, by having $2m$ tuples of random scalars in the secret key, and a global value θ for the encryption. Following the techniques from [ABB+13], and the work from [EHK+13] this scheme is `VIND-PO-CCA` under the MDDH assumption.

k-linear Smooth Projective Hash Function

One can now build a Hash Proof system on this CCA-2 scheme, by following [EHK+13, BBC+13b]

[11] Like Cramer-Shoup one could rely on an universal one-way hash function family instead

– $\mathsf{HashKG}(\mathfrak{L}_M) : \mathsf{hk} \xleftarrow{\$} \mathbb{Z}_p^{k+2}$,
– $\mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}_M, \ell, [\mathcal{C}]_2)) :$ Setting $\mathsf{ht} = \begin{matrix}\mathsf{pk}_1 \\ \mathsf{pk}_2 + \theta\mathsf{pk}_3\end{matrix}$, we have $[\mathsf{hp}]_2 = \left[\mathsf{hk}^\top \mathsf{ht}\right]_2$, where $\theta = H(\ell, \boldsymbol{e})$,
– $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}_M, \ell, [\mathcal{C}]_2)) : [H]_2 \leftarrow \left[\mathsf{hk}^\top \left(\mathcal{C} - (0 \mid M)^\top\right)\right]_2$,
– $\mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}_M, \ell, [\mathcal{C}]_2), \boldsymbol{\mu})[H']_2 \leftarrow [\mathsf{hp}\boldsymbol{\mu}]_2$

The smoothness of such system is show by considering the determinant of the matrix $\boldsymbol{D} = \begin{bmatrix}\mathsf{ht}^\top \\ \boldsymbol{C} - (0 \mid M)^\top\end{bmatrix}$

As soon as $\boldsymbol{C}$ is not a valid encryption of M, this matrix has non zero determinant. As the first lines leads to $\mathsf{hk}^\top\mathsf{ht}$ the public projection key hp, and the last to $\mathsf{hk}^\top(\boldsymbol{C} - (0 \mid M))$ the computed Hash value, we can deduce that form the public views, the computed hash value is information theoretically independent from the projection keys for words outside the language.

C Proof of the Security of Fragmented IBE

In this section, we briefly prove that the variation over the [BKP14] IBE, does not weaken it's security.

Theorem 18. *The Blinded IBE achieves the leak-free secret key generation requirements under the security of the initial IBE, the extractability of the* MDDH *Cramer Shoup Encryption and the Smoothness of the SPHF on the bit commitment.*

Proof. Given an adversary $\mathcal{A}$ against the security of the blinded scheme, we are going to build an adversary $\mathcal{B}$ against the security of the initial IBE.

From the challenge, $\mathcal{B}$ receives the parameter from an IBE scheme mpk, has access to a user key generation oracle $\mathsf{USKGen}_{\mathcal{O}}$, and for a given fresh id^*, a tuple $(\mathsf{K}^*, \mathsf{C}^*)$ whose consistency he need to decide.

In the initial game $\mathcal{G}_0$, $\mathcal{B}$ behaves normally, generating $\mathsf{mpk}, \mathsf{msk}$, and answering $\mathcal{A}$ blinded request honestly.

$\mathcal{G}_1$ In this game, $\mathcal{B}$ starts altering his answer. Using the extraction procedure on the CCA commitment, $\mathcal{B}$ is able to recover the identity id queried by $\mathcal{A}$. If for a bit i, there is two valid openings, then $\mathcal{A}$ broke the collision resistance property of the underlying chameleon hash (and so MDDH) and $\mathcal{B}$ aborts. The Chameleon hash being Collision Resistant, this game is equivalent to the previous one under $k - \mathsf{MDDH}$.

$\mathcal{G}_2$ In this game, $\mathcal{B}$ starts altering his answer. Using the extraction procedure on the CCA commitment, $\mathcal{B}$ is able to recover the identity id queried by $\mathcal{A}$. Now, for each bit, there is at least one dummy value $\bar{\mathsf{id}}_i$, and so $\mathcal{B}$ computes random values $\boldsymbol{\omega}_{i,\bar{\mathsf{id}}_i} \xleftarrow{\$} \mathbb{Z}_p^{k+1}$. Under the smoothness of the underlying smooth projective hash function, this game is indistinguishable from the previous one.

$\mathcal{G}_3$ Now $\mathcal{B}$ continues to extract the requested id, picks random $\boldsymbol{f} \xleftarrow{\$} \mathbb{Z}_p^{\ell \times k+1}$, and sets $\boldsymbol{w}_0 = \mathsf{usk}[\mathsf{id}] - \sum_{i=1}^{\ell} \boldsymbol{f}_i$, then for $i \in [\![1, n]\!]$, $\boldsymbol{\omega}_{i,\mathsf{id}_i} = \boldsymbol{f}_i + H_{i,\mathsf{id}_i}$ while $\boldsymbol{\omega} i, \mathsf{id}_i \xleftarrow{\$} \mathbb{Z}_p^{k+1}$ as before. If $\mathcal{B}$, does not recover a valid identity, he simply only sends dummy values.
This game is indistinguishable from the previous one, as this is just a rewriting of the vector $\boldsymbol{f}_i$. (Noting $\hat{\boldsymbol{f}}_i$ the old one, we have $\boldsymbol{f}_i = \hat{\boldsymbol{f}}_i - (\mathsf{id}_i \boldsymbol{Y}_i)\boldsymbol{t}$ which follows the same distribution).

$\mathcal{G}_4$ $\mathcal{B}$ can now forget about msk, when receiving a $\mathsf{BlindUSKGen}$ request, $\mathcal{B}$ extracts the request identity, and if it is not $\perp$ he forwards it to the $\mathsf{USKGen}_{\mathcal{O}}$ oracle, and plugs the received $\mathsf{usk}[\mathsf{id}]$ as before.

Now $\mathcal{A}$ can request a challenge from $\mathcal{B}$, $\mathcal{B}$ forwards this request to the initial non-blinded IBE challenge for a fresh id^*, and returns the challenge $(\mathsf{K}^*, \mathsf{C}^*)$ to $\mathcal{A}$ which leads to the conclusion. □

D Proof of the Generic Construction of Adaptive Oblivious Transfer

We prove the security of this protocol via a sequence of games, starting from the real game, where the adversary $\mathscr{A}$ interacts with the real players, and ending with the ideal game, where we have built a simulator $\mathscr{S}$ that makes the interface between the ideal functionality $\mathcal{F}$ and the adversary $\mathscr{A}$. Recall that we consider adaptive corruptions.

We denote as $\mathcal{S}$ the server and $\mathcal{U}$ the user. The main idea is that, by assumption, the simulator can always obtain the common trapdoor tk of the collection of languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ and use it to commit to a word simultaneously belonging to all the languages. In case of adaptive corruption, we face two cases: either the word was not correct, in which case, following the real protocol, the user should have erased his randomness, so that the simulator does not have anything to reveal. Or the word was correct and should belong to a certain language $\mathfrak{L}_s$, and the simulator can then adapt the word and randomness so that it seems to belong to $\mathfrak{L}_s$ (only). This enables us to avoid the use of commitments both extractable and equivocable (which is the usual tool for adaptive corruptions).

Due to the construction of the protocol, we have to prove that the user recovers the secret key $\mathsf{usk}[s]$ corresponding to the s-th line of the database in an oblivious way, which means on the one hand that the user gains no information on the other keys, and on the other hand that the server gains no information on the key required by the user. Assuming this is the case (the proof follows), the adaptive security of the global oblivious transfer relies on the security of the underlying IBE scheme: The indistinguishability of the ciphertexts ensure that the user only recovers the s-th line for which he knows the secret key $\mathsf{usk}[s]$.

Since the channels are authenticated, we know whether a flow was sent by an honest player (and received without any alteration) or not.

Game G_0: This is the real game.

Game G_1: In this game, the simulator generates correctly every flow on behalf of the honest players, as they would do themselves, knowing the database $(DB(1), \ldots, DB(n))$ and the word W sent by the environment to the server and the user. In all the subsequent games, the players use the label $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{S}, \mathcal{U})$. In case of corruption, the simulator can give the internal data generated on behalf of the honest users.

Game G_2: In this game, we replace the setup algorithm Setup by SetupT, allowing the existence of a trapdoor to find words in the intersection of the collection of languages. We also allow the simulator to program $\mathsf{Setup}_{\mathsf{CCA}}$ in the CRS, enabling it to learn the extraction trapdoor of the CCA encryption scheme. The indistinguishability of the setups makes this game indistinguishable from the former one for the environment. Corruptions are handled the same way.

Game G_3: We first deal with honest servers $\mathcal{S}$: he computes everything honestly during the database preparation. When receiving a commitment C, the simulator extracts the committed value W. By testing with the help of the algorithm Verify, it recovers s such that $W \in \mathfrak{L}_s$. If it recovers $s \neq t$ such that $W \in \mathfrak{L}_s \cap \mathfrak{L}_t$, then the adversary has broken the infeasibility of finding a word in an intersection of languages without knowing the trapdoor and we abort the game. Otherwise, instead of computing the keys $H_{i,b}$, for $i = 1, \ldots, \ell$ and $b = 0, 1$ with the hash function, the simulator then chooses $H_{i,b} \xleftarrow{\$} G$ when b is not equal to the i-th bit of W.

With an hybrid proof, applying the smoothness for every honest sender, on every index (i, b) such that $b \neq W_i$, since C is extracted to $W \in \mathfrak{L}_i$, for any (i, b) such that $b \neq W_i$, the hash value is indistinguishable from a random value.

In case of corruption, everything has been erased (except after the pre-flow, where the simulator can reveal the keys $(\mathsf{pk}, \mathsf{sk})$ generated honestly). This game is thus indistinguishable from the previous one under the smoothness.

Game G_4: Still in this case, when receiving a commitment C, the simulator extracts the committed value W. By testing with the help of the algorithm Verify, it recovers s such that $W \in \mathfrak{L}_s$. Instead of proceeding as the server would do with $(\mathsf{usk}[i, b])$, the simulator proceeds on $(\mathsf{usk}'[i, b])$, with $\mathsf{usk}'[i, b] = 0$ except if $b = W_i$. Since the masks $H_{i,b}$, for $b \neq W_i$, are random, this game is perfectly indistinguishable from the previous one.

Game G_5: We now deal with honest users $\mathcal{U}$: the simulator now uses the trapdoor tk to find a word W' in the intersection of all languages.

With an hybrid proof, applying a security game in each session in which the simulator does not know the trapdoor tk, one can show the indistinguishability of the two games. In case of corruption of the receiver, one learns the already known value W, thus s.

Game G_6: We deal with **the generation of R for honest servers $\mathcal{S}$ where the users $\mathcal{U}$ are honests**: if $\mathcal{S}$ and $\mathcal{U}$ are honest at least until $\mathcal{S}$ received the second flow, the simulator sets $R = F(S')$ for both $\mathcal{S}$ and $\mathcal{U}$, with S' a random value, instead of $R = F(S)$.

With an hybrid proof, applying the IND-CPA property for each session, one can show the indistinguishability of this game with the previous one.

Game G_7: Still in the same case, the simulator sets R as a random value, instead of $R = F(S')$.

With an hybrid proof, applying the PRF property for each session, one can show the indistinguishability of this game with the previous one.

Game G_8: We now deal with **the generation of H_{i,W_i} for honest servers $\mathcal{S}$ with honest users $\mathcal{U}$.** Thanks to the additional random mask R, one can send random $(\mathsf{usk}[i, W_i])_i$, and H_{i,W_i} can be computed later (when $\mathcal{U}$ actually receives its flow).

As above, but only if $\mathcal{U}$ has not been corrupted before receiving its flow, the simulator chooses $H_{i,W_i} \xleftarrow{\$} G$. With an hybrid proof, applying the pseudo-randomness, for every honest sender, the hash value is indistinguishable from a random value, because the adversary does not know any decommitment information for C. If the player $\mathcal{U}$ involved in the pseudo-randomness game gets corrupted (but the decommitment information is unknown) we are not in this case, and we can thus abort it.

In case of corruption of $\mathcal{S}$, everything has been erased (except after the pre-flow, where the simulator can reveal the keys $(\mathsf{pk}, \mathsf{sk})$ generated honestly).

In case of corruption of the receiver $\mathcal{U}$, and thus receiving the value $\widetilde{DB(s)}$, the simulator computes $\tilde{K}_s$ such that $\widetilde{K_s} \oplus \widetilde{DB(s)} = K_s \oplus DB(s)$ and the corresponding $\widetilde{\mathsf{usk}[W]}$. It chooses R (because it was a random value unknown to the adversary and all the other $H_{i,b}$ are independent random values too) such that $\left(\bigoplus_i \mathsf{busk}[i, W_i] \oplus H'_{i,W_i}\right) \oplus Z \oplus R = \widetilde{\mathsf{usk}[W]}$.

This game is thus indistinguishable from the previous one under the pseudo-randomness.

Remark. We now explain how, in the pairing instanciation of our protocol, given already sent values $D_s, K_s \oplus DB(s)$, a simulator recovering from the environment the value $\widetilde{DB(s)}$, can adaptively be able to change his memory so as to compute a user key $\widetilde{\mathsf{usk}[W]}$ such that $\mathsf{Dec}(\widetilde{\mathsf{usk}[W]}, W, D_s) = DB(s) \oplus K_s \oplus \widetilde{DB(s)}$. This is exactly where we use the restriction on the size of DB elements so that we can manage to find a vector $\delta_s \in \mathbb{G}_2^{2k+1}$ such that $DB(s) \oplus \widetilde{DB(s)} = e(D_s, \delta_s)$. Thus, this allows the server to update his memory into $\widetilde{\mathsf{usk}[W]} = \mathsf{usk}[W] \cdot \delta_s$.

Game G_9: Still in this case, the simulator proceeds on $(\mathsf{usk}[i, b])$, with $\mathsf{usk}[i, b] = 0$ for all i, b. Since the masks $H_{i,b}$, $Z \oplus R$, for any i, b, are independent random values (the $\mathsf{busk}[i, b]$, for $b \neq W_i$ are independent random values, and R is independently random), this game is perfectly indistinguishable from the previous one.

We remark that it is therefore no more necessary to know the index s given by the ideal functionality to the honest receiver $\mathcal{U}$, to simulate $\mathcal{S}$ (but it is still necessary to simulate $\mathcal{U}$).

Game G_{10}: We do not use anymore the knowledge of s when simulating an **honest user** $\mathcal{U}$: the simulator generates a word W' in the intersection of the languages and $C \xleftarrow{\$} \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(W'; \boldsymbol{\rho})$, with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{S}, \mathcal{R})$, to send C during the first phase of honest users. This does not change anything from the previous game since the randomness needed to open to a word in another language is never revealed.

When it thereafter receives $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{S}, \mathcal{R}, (\mathsf{hp}_{i,b}, \mathsf{busk}[i,b]))$ from the adversary, the simulator computes, for all lines t, $\mathsf{usk}[W_t]$ and recovers K_t and finally $DB(t)$, which provides the database $(DB(1), \ldots, DB(n))$ submitted by the sender. It uses them to send a `Send`-message to the ideal functionality.

Game G_{11}: We can now make use of the functionality, which leads to the following simulator:

- when receiving a `Send`-message from the ideal functionality, which means that an honest server has sent a pre-flow and a database, the simulator generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^{\mathfrak{K}})$ and sends pk as pre-flow;
- after receiving a pre-flow pk (from an honest or a corrupted sender) and a `Receive`-message from the ideal functionality, which means that an honest receiver has sent a first flow, the simulator generates a word W' in the intersection of languages, $C \xleftarrow{\$} \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(W; \boldsymbol{\rho})$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{R}, \mathcal{S})$ and $c \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{pk}, S)$ where S is a random value;
- when receiving a commitment C and a ciphertext c, generated by the adversary (from a corrupted receiver), the simulator extracts the committed value W and recovers s (aborting in case of multiple values), and uses it to send a `Receive`-message to the ideal functionality (and also decrypts the ciphertext c as S, and computes $R = F(S)$);
- when receiving $(\mathsf{hp}_{i,b}, \mathsf{busk}[i,b])$ from the adversary, the simulator computes, for all lines t, $\mathsf{usk}[W_t]$ and recovers K_t and finally $DB(t)$, which provides the database $(DB(1), \ldots, DB(n))$ submitted by the sender. It uses them to send a `Send`-message to the ideal functionality.
- when receiving a `Received`-message from the ideal functionality, together with $\widetilde{DB(s)}$, on behalf of a corrupted receiver, from the extracted W leading to s, instead of proceeding as the sender would do on $(\mathsf{usk}[i,b])$, the simulator proceeds on $(\mathsf{usk}'_{[}i,b])$, with $\mathsf{usk}'_{[}i,b] = 0$ except if $b = W_i$.
- when receiving a commitment C and a ciphertext c, generated by an honest sender (*i.e.*, by the simulator itself), the simulator proceeds as above on $(\mathsf{usk}'_{[}i,b])$, with $\mathsf{usk}'_{[}i,b] = 0$ except if $b = W_i$, but it chooses R uniformly at random instead of choosing it as $R = F(S)$; in case of corruption afterward, the simulator will adapt R such that $\left(\bigoplus_i \mathsf{busk}[i, W_i] \oplus H'_{i,W_i}\right) \oplus Z \oplus R = \widetilde{\mathsf{usk}[W]}$, where $\widetilde{\mathsf{usk}[W]}$ leads to $\tilde{K}_s$ such that $\widetilde{K_s} \oplus \widetilde{DB(s)} = K_s \oplus DB(s)$, where $\widetilde{DB(s)}$ is the message actually received by the receiver.

Any corruption either reveals s earlier, which allows a correct simulation of the receiver, or reveals $(DB(1), \ldots, DB(n))$ earlier, which allows a correct

simulation of the server. When the server has sent his flow, he has already erased all his random coins.

However, there would have been an issue when the user is corrupted after the server has sent is flow, but before the user receives it, since he has kept $\boldsymbol{\rho}$: this would enable the adversary to recover $\mathsf{usk}[W]$ from $\mathsf{busk}[i, W_i]$ and hp_{i,W_i}. This is the goal of the ephemeral mask R that provides a secure channel.

E Proof of the Generic Construction of Oblivious Language-Based Envelope

We prove the adaptive[12] security of this protocol via a sequence of games, starting from the real game, where the adversary $\mathscr{A}$ interacts with the real players, and ending with the ideal game, where we have built a simulator $\mathscr{S}$ that makes the interface between the ideal functionality $\mathcal{F}$ and the adversary $\mathscr{A}$.

We denote as $\mathcal{S}$ the sender (*i.e.* the server) and $\mathcal{R}$ the receiver (*i.e.* the user). The main idea is that, by assumption, the simulator can always obtain the common trapdoor tk of the collection of languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ and use it to commit to a word simultaneously belonging to all the languages. In case of adaptive corruption, we face two cases: either the word was not correct, in which case, following the real protocol, the user should have erased his randomness, so that the simulator does not have anything to reveal. Or the word was correct and should belong to a certain language $\mathfrak{L}_i$, and the simulator can then adapt the word and randomness so that it seems to belong to $\mathfrak{L}_i$ (only). This enables us to avoid the use of commitments both extractable and equivocable (which is the usual tool for adaptive corruptions).

We say that a flow is *oracle-generated* if it was sent by an honest player (or the simulator) and received without any alteration by the adversary. It is said *non-oracle-generated* otherwise.

Game G_1: This is the real game.

Game G_2: In this game, the simulator generates correctly every flow on behalf of the honest players, as they would do themselves, knowing the inputs $(P_1, \ldots, P_n)$ and W sent by the environment to the sender and the receiver. In all the subsequent games, the players use the label $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{S}, \mathcal{R})$. In case of corruption, the simulator can give the internal data generated on behalf of the honest players.

Game G_3: In this game, we replace the setup algorithm Setup by SetupT, allowing the existence of a trapdoor to find words in the intersection of the collection of languages. We also allow the simulator to program $\mathsf{Setup}_{\mathsf{CCA}}$ in the CRS, enabling it to learn the extraction trapdoor of the CCA encryption scheme. The indistinguishability of the setup makes this game indistinguishable from the former one for the environment. Corruptions are handled the same way.

[12] One can obtain the proof of the static version by removing the parts related to the pre-flow and to the adaptive corruptions in the proof below.

Game G_4: We first deal with oracle-generated flows from the senders $\mathcal{S}$: when receiving a commitment C, the simulator extracts the committed value W. By testing with the help of the algorithm Verify, it recovers i such that $W \in \mathfrak{L}_i$. If it recovers $i \neq j$ such that $W \in \mathfrak{L}_i \cap \mathfrak{L}_j$, then the adversary has broken the infeasibility of finding a word in an intersection of languages without knowing the trapdoor and we abort the game. Otherwise, instead of computing the key v_t, for $t = 1, \ldots, n$ with the hash function, the simulator then chooses $v_t \xleftarrow{\$} G$ for $t \neq i$.

With an hybrid proof, applying the smoothness for every honest sender, on every index $t \neq i$, since C is extracted to $W \in \mathfrak{L}_i$, for any $t \neq i$, the hash value is indistinguishable from a random value.

In case of corruption, everything has been erased (except after the pre-flow, where the simulator can reveal the keys $(\mathsf{pk}, \mathsf{sk})$ generated honestly). This game is thus indistinguishable from the previous one under the smoothness.

Game G_5: Still in this case, when receiving a commitment C, the simulator extracts the committed value W, giving it the number i. Instead of proceeding as the sender would do on $(P_1, \ldots, P_n)$, the simulator proceeds on $(P'_1, \ldots, P'_n)$, with $P'_i = P_i$, but $P'_t = 0$ for all $t \neq i$. Since the masks v_t, for $t \neq i$, are random, this game is perfectly indistinguishable from the previous one.

Game G_6: We now deal with oracle-generated flows from the receivers $\mathcal{R}$: the simulator now uses the trapdoor tk to find a word W in the intersection of all languages.

With an hybrid proof, applying a security game in each session in which the simulator does not know the trapdoor tk, one can show the indistinguishability of the two games. In case of corruption of the receiver, one learns the already known value i.

Game G_7: We deal with **the generation of R for honest senders $\mathcal{S}$ on oracle-generated queries (adaptive case only)**: if $\mathcal{S}$ and $\mathcal{R}$ are honest at least until $\mathcal{S}$ received the second flow, the simulator sets $R = F(J')$ for both $\mathcal{S}$ and $\mathcal{R}$, with J' a random value, instead of $R = F(J)$.

With an hybrid proof, applying the $\mathsf{IND\text{-}CPA}$ property for each session, one can show the indistinguishability of this game with the previous one.

Game G_8: Still in the same case, the simulator sets R as a random value, instead of $R = F(J')$.

With an hybrid proof, applying the PRF property for each session, one can show the indistinguishability of this game with the previous one.

Game G_9: We now deal with **the generation of v_i for honest senders $\mathcal{S}$ on oracle-generated queries**:

- in the static case (the pre-flow is only needed to compute $(\mathsf{vk}, \mathsf{vtk})$, and thus we assume $R = 0$) the simulator chooses $v_i \xleftarrow{\$} G$ (for $t \neq i$, the simulator already chooses $v_t \xleftarrow{\$} G$), where i is the index given by the ideal functionality to the honest receiver $\mathcal{R}$.

With an hybrid proof, applying the pseudo-randomness for every honest sender, the hash value is indistinguishable from a random value, because the adversary does not know any decommitment information for C;

– in the adaptive case, and thus with the additional random mask R, one can send a random P_i, and v_i can be computed later (when $\mathcal{R}$ actually receives its flow).
As above, but only if $\mathcal{R}$ has not been corrupted before receiving its flow, the simulator chooses $v_s \xleftarrow{\$} G$. With an hybrid proof, applying the pseudo-randomness, for every honest sender, the hash value is indistinguishable from a random value, because the adversary does not know any decommitment information for C. If the player $\mathcal{R}$ involved in the pseudo-randomness game gets corrupted (but the decommitment information is unknown) we are not in this case, and we can thus abort it.
In case of corruption of $\mathcal{S}$, everything has been erased (except after the pre-flow, where the simulator can reveal the keys $(\mathsf{pk}, \mathsf{sk})$ generated honestly).
In case of corruption of the receiver $\mathcal{R}$, and thus receiving the value P_i, the simulator chooses R (because it was a random value unknown to the adversary and all the other v_t are independent random values too) such that

$$R \oplus \mathsf{ProjHash}(\mathsf{hp}_i, (L_i, \mathsf{param}), (\ell, C), r_i) \oplus P_i = Q_i.$$

This game is thus indistinguishable from the previous one under the pseudo-randomness.

Game G_{10}: Still in this case, the simulator proceeds on $(P'_1, \ldots, P'_n)$, with $P'_t = 0$ for all i. Since the masks $v_t \oplus R$, for any $t = 1, \ldots, n$, are independent random values (the v_t, for $t \neq i$ are independent random values, and v_i is also independently random in the static case, while R is independently random in the adaptive case), this game is perfectly indistinguishable from the previous one.

We remark that it is therefore no more necessary to know the index i given by the ideal functionality to the honest receiver $\mathcal{R}$, to simulate $\mathcal{S}$ (but it is still necessary to simulate $\mathcal{R}$).

Game G_{11}: We do not use anymore the knowledge of i when simulating an **honest receiver** $\mathcal{R}$: the simulator generates a word W in the intersection of the languages and $C \xleftarrow{\$} \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(W; r)$, with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{S}, \mathcal{R})$, to send C during the first phase of honest receivers. This does not change anything from the previous game since the randomness needed to open to a word in another language is never revealed.

When it thereafter receives $(\mathsf{Send}, \mathsf{sid}, \mathsf{ssid}, \mathcal{S}, \mathcal{R}, (Q_1, \mathsf{hp}_1, \ldots, Q_n, \mathsf{hp}_n))$ from the adversary, the simulator computes, for $i = 1, \ldots, k$, r_i,

$$v_i \leftarrow \mathsf{ProjHash}(\mathsf{hp}_i, (\mathfrak{L}_i, \mathsf{param}), (\ell, C), r_i)$$

and $P_i = v_i \oplus R \oplus Q_i$. This provides the database submitted by the sender.

Game G_{12}: We can now make use of the functionality, which leads to the following simulator:

– when receiving a `Send`-message from the ideal functionality, which means that an honest sender has sent a pre-flow, the simulator generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^{\mathfrak{K}})$ and sends pk as pre-flow;
– after receiving a pre-flow pk (from an honest or a corrupted sender) and a `Receive`-message from the ideal functionality, which means that an honest receiver has sent a first flow, the simulator generates a word W in the intersection of languages, $C \xleftarrow{\$} \mathsf{Encrypt}^{\ell}_{\mathrm{cca}}(W; r)$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{R}, \mathcal{S})$ and $c \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{pk}, J)$ where R is a random value;
– when receiving a commitment C and a ciphertext c, generated by the adversary (from a corrupted receiver), the simulator extracts the committed value W and recovers i (aborting in case of multiple values), and uses it to send a `Receive`-message to the ideal functionality (and also decrypts the ciphertext c as J, and computes $R = F(J)$);
– when receiving $(Q_1, \mathsf{hp}_1, \ldots, Q_n, \mathsf{hp}_n)$ from the adversary (a corrupted sender), the simulator computes, for $i = 1, \ldots, n$, r_i,

$$v_i \leftarrow \mathsf{ProjHash}(\mathsf{hp}_i, (\mathfrak{L}_i, \mathsf{param}), (\ell, C), r_i)$$

and $P_i = v_i \oplus R \oplus Q_i$. It uses them to send a `Send`-message to the ideal functionality.
– when receiving a `Received`-message from the ideal functionality, together with P_i, on behalf of a corrupted receiver, from the extracted W leading to i, instead of proceeding as the sender would do on $(P_1, \ldots, P_n)$, the simulator proceeds on $(P'_1, \ldots, P'_n)$, with $P'_i = P_i$, but $P'_j = 0$ for all $j \neq i$;
– when receiving a commitment C and a ciphertext c, generated by an honest sender (*i.e.*, by the simulator itself), the simulator proceeds as above on $(P'_1, \ldots, P'_n)$, with $P'_j = 0$ for all j, but it chooses R uniformly at random instead of choosing it as $R = F(J)$; in case of corruption afterward, the simulator will adapt R such that $R \oplus \mathsf{ProjHash}(\mathsf{hp}_i, (\mathfrak{L}_i, \mathsf{param}), (\ell, C), r_i) \oplus Q_i = P_i$, where P_i is the message actually received by the receiver.

Any corruption either reveals i earlier, which allows a correct simulation of the receiver, or reveals $(P_1, \ldots, P_n)$ earlier, which allows a correct simulation of the sender. When the sender has sent his flow, he has already erased all his random coins.

However, there would have been an issue when the receiver is corrupted after the sender has sent is flow, but before the receiver receives it, since he has kept r_i: this would enable the adversary to recover P_i from Q_i and hp_i. This is the goal of the ephemeral mask R that provides a secure channel.

F Oblivious Primitives Obtained by the Framework

The framework presented in Section 4 page 20 provides a generic way to achieve asymmetric protocols around automated trust negotiation. In this section, we show that the classical oblivious primitives directly lie in this framework and can be seen as examples of Oblivious Language-Based Envelope. We in particular show how by tweaking the languages (and choosing appropriate smooth

projective hash functions), one can achieve current instantiations of Oblivious Signature-Based Envelope or Oblivious Transfer. We also show how to give a new instantiation of Access Controlled Oblivious Transfer under classical assumptions.

For the specific instantiations, without loss of generality, we are going to focus on elliptic cryptography, but as shown in the case of Oblivious Transfer in [BC15], most building blocks behave the same way under other families of assumptions. As we focus on elliptic curve instantiations, we are going to use ElGamal encryption as the CPA encryption used for the pre-flow.

F.1 Oblivious Signature-Based Envelope

Ideal Functionality for Oblivious Signature-Based Envelope. In order to obtain OSBE as a special case of OLBE, we assume $n = n_{\max} = 1$ and we consider the inner language $\mathfrak{L}$ of valid signatures of a public message M under a given public verification key vk. The corresponding private signing key sk is the language secret key $\mathsf{sk}_{\mathfrak{L}}$, and we assume the existence of a trapdoor key $\mathsf{tk}_{\mathfrak{L}}$ which allows to sample new signatures without the knowledge of $\mathsf{sk}_{\mathfrak{L}}$[13]. Plugging these values in the previous framework directly gives the ideal functionality $\mathcal{F}_{\mathrm{OSBE}}$ in the Simple UC framework, presented in Figure 11, which has never been explicitly given before, to the best of our knowledge.

The functionality $\mathcal{F}_{\mathrm{OSBE}}$ is parametrized by a security parameter $\mathfrak{K}$ and a signature scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. It interacts with an adversary $\mathscr{S}$ and a set of parties $\mathfrak{P}_1, \ldots, \mathfrak{P}_N$ via the following queries:

- **Upon receiving an input $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, P)$ from party $\mathfrak{P}_i$**, with $P \in \{0,1\}^{\mathfrak{K}}$: record the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, P)$ and reveal $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$. Ignore further $\mathtt{Send}$-message with the same ssid from $\mathfrak{P}_i$.
- **Upon receiving an input $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, \sigma)$ from party $\mathfrak{P}_j$**: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, P)$ is not recorded. Otherwise, reveal $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$ and send $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, P')$ to $\mathfrak{P}_j$ where $P' = P$ if $\mathsf{Verify}(\mathsf{vk}, \sigma, M)$ returns 1, and $P' = \perp$ otherwise. Ignore further $\mathtt{Receive}$-message with the same ssid from $\mathfrak{P}_j$.

Fig. 11. Ideal Functionality for Oblivious Signature-Based Envelope $\mathcal{F}_{\mathrm{OSBE}}$

Generic Construction and Pairing-Based Instantiation. We give in the left part of Figure 12 the building blocks for OSBE, which directly give a generic construction for OSBE by plugging them into our generic construction for OLBE in Section 4.4.

[13] Note that $\mathsf{tk}_{\mathfrak{L}}$ may not directly lead to sk but possibly to an alternative signing algorithm.

	Generic OSBE	Instantiation Proposal
	Signature and inner language	Waters Signature
	sk, vk	h^x, g^x
$\sigma(m) =$	$\mathsf{Sign}(\mathsf{sk}, m)$	$h^x(u_0 \prod u_i^{m_i})^s, g^s$
$\mathfrak{L}_m =$	$\{\sigma \mid \mathsf{Verify}(\mathsf{vk}, \sigma, m) = 1\}$	$\{\boldsymbol{\sigma} \mid e(\sigma_1, g^x) \cdot e(\sigma_2, F(m)) = e(h, g^x)\}$
$\mathsf{sk}_{\mathfrak{L}} =$	sk	h^x
$\mathsf{tk}_{\mathfrak{L}} =$	allows to compute a new signature	$\log_g h$
	Compatible CCA Encryption	Linear Cramer Shoup Encryption
$\mathsf{ek} =$	ek	$(h, u, v, w, c_1, d_1, c_2, d_2)$
$\mathcal{C} =$	$\mathsf{Encrypt}(\mathsf{ek}, \sigma; \rho)$	$h^{r+t} \cdot h^x F(m)^s, u^r, v^t, w^{r+t},$ $(c_1 d_1^\theta)^r (c_2 d_2^\theta)^t, g^s$ where $\theta = \mathcal{H}(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4)$
	SPHF	SPHF
$\mathsf{hk} =$	$\mathsf{HashKG}(\mathsf{ek}, \mathfrak{L}_m)$	$\alpha, \beta, \lambda, \mu, \nu \xleftarrow{\$} \mathbb{Z}_p$
$\mathsf{hp} =$	$\mathsf{ProjKG}(\mathsf{hk}, \mathsf{ek}, \mathfrak{L}_m, \mathcal{C})$	$h^\alpha u^\beta w^\mu (c_1 d_1^\theta)^\nu, h^\alpha v^\lambda w^\mu (c_2 d_2^\theta)^\nu$
$H =$	$\mathsf{Hash}(\mathsf{hk}, \mathsf{ek}, \mathfrak{L}_m, \mathcal{C})$	$e(\mathcal{C}_2, g)^\beta e(\mathcal{C}_3, g)^\lambda e(\mathcal{C}_4, g)^\mu e(\mathcal{C}_5, g)^\nu$ $(e(\mathcal{C}_1, g)/(e(h, g^x) e(F(M), \mathcal{C}_6)))^\alpha$
$H' =$	$\mathsf{ProjHash}(\mathsf{hp}, \mathsf{ek}, \mathfrak{L}_m, \rho)$	$e(\mathsf{hp}_1^r \cdot \mathsf{hp}_2^t, g)$

Fig. 12. Setting the Language to instantiate an OSBE, SPHF is based on [BPV12]

Instantiating them with Waters' signature scheme [Wat05] and Linear Cramer Shoup encryption [CKP07, Sha07], it gives us a first pairing-based instantiation for OSBE, depicted in the right part of Figure 12, secure in the UC framework against adaptive corruptions (assuming reliable erasures).

Interestingly, this scheme ends up being very similar to the one presented in [BPV12] and proven secure in the standard (not UC) security model, which shows that their scheme remains indeed secure in the UC framework. What was lacking in their paper was the existence (and knowledge) of the trapdoor $\mathsf{tk}_{\mathfrak{L}}$ which, as we can see in the generic proof for OLBE (Appendix E), allows the simulator to be able to always send a valid signature in the first flow on behalf of the receiver. This enables it to be prepared to face adaptive corruptions, whatever input the environment gives it later on.

F.2 1-out-of-n Oblivious Transfer

Ideal Functionality for 1-out-of-n Oblivious Transfer. The ideal functionality of a 1-out-of-n Oblivious Transfer (OT) protocol, from [Can01, CKWZ13, ABB+13], is depicted in Figure 13 (adapted to the Simple UC framework). As explained in Section 3.1 page 11, one can easily see it as a special application of our generic OLBE ideal functionality, considering $n \geq 2$ and $n_{\max} = 1$. Indeed, the only change we have to make is not to consider anymore the line numbers as simple numbers, but to "encode" them (the exact encoding will depend on the protocol). For every line i, the language $\mathfrak{L}_i$ will correspond to a representation of line i. Instead of directly giving s to the functionality, the receiver will give it a word $W_s \in \mathfrak{L}_s$. This leads to the functionality given in Figure 2 page 12.

The functionality $\mathcal{F}_{(1,n)\text{-OT}}$ is parametrized by a security parameter $\mathfrak{K}$. It interacts with an adversary $\mathscr{S}$ and a set of parties $\mathfrak{P}_1,\ldots,\mathfrak{P}_N$ via the following queries:

- **Upon receiving an input $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (m_1, \ldots, m_n))$ from $\mathfrak{P}_i$,** with $m_k \in \{0,1\}^{\mathfrak{K}}$: record the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (m_1, \ldots, m_n))$ and reveal $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to $\mathscr{S}$. Ignore further `Send`-message with the same ssid from $\mathfrak{P}_i$.
- **Upon receiving an input $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, s)$ from $\mathfrak{P}_j$,** with $s \in \{1,\ldots,n\}$: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (m_1, \ldots, m_n))$ is not recorded; otherwise reveal $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to $\mathscr{S}$, send $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, m_s)$ to $\mathfrak{P}_j$ and ignore further `Receive`-message with the same ssid from $\mathfrak{P}_j$.

Fig. 13. Ideal Functionality for 1-out-of-n Oblivious Transfer $\boldsymbol{\mathcal{F}_{(1,n)\text{-OT}}}$

	Generic 1-out-of-n Oblivious Transfer	Possible Instantiation [ABB+13]
	Inner language	Chameleon Hashed Numbers
	Verifiable Chameleon Hash Keys $(\mathsf{ck}, \mathsf{tk}, \mathsf{vtk})$	$(g_1, g \xleftarrow{\$} \mathbb{G}_2), \alpha = \log_g g_1$,
$c =$	$\mathsf{Commit}_{\mathsf{ck},\mathcal{V}}(s)$	$\forall i$ such that $s = \|_{i=1}^{\log n} s_i$: $a_i = g_1^{r_{i,s_i}} g^{s_i}, \boldsymbol{r}_i = (r_{i,0}, r_{i,1})$
$\mathfrak{L}_s =$	$\{c \mid \mathsf{Verify}(\mathsf{vtk}, c, s) = 1\}$	$\{(\boldsymbol{a}, \boldsymbol{r}) \mid \forall i, a_i = g_1^{r_{i,s_i}} g^{s_i}\}$
$\mathsf{sk}_{\mathfrak{L}} =$	$\perp$	$\perp$
$\mathsf{tk}_{\mathfrak{L}} =$	tk	$\log_g h$
	Compatible CCA Encryption	Cramer Shoup Encryption
$\mathsf{ek} =$	ek	$h, u, v, c, d, f \xleftarrow{\$} \mathbb{G}_1$
$\mathcal{C} =$	$\mathsf{Encrypt}(\mathsf{ek}, c; \rho)$	$\forall i, b, (h^{\rho_{i,b}} \cdot h^x f^{r_{i,b}}, u^{\rho_{i,b}}, v^{s_{\rho,b}}, (cd^\theta)^{s_{\rho,b}}, a_i)$ where $\theta = \mathcal{H}(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$
	SPHF	SPHF
$\mathsf{hk}_s =$	$\mathsf{HashKG}(\mathsf{ek}, \mathfrak{L}_s)$	$\alpha_s, \beta_s, \mu_s, \nu_s \xleftarrow{\$} \mathbb{Z}_p$
$\mathsf{hp}_s =$	$\mathsf{ProjKG}(\mathsf{hk}_s, \mathsf{ek}, \mathfrak{L}_s, \mathcal{C})$	$h^{\alpha_s} u^{\beta_s} v^{\mu_s} (cd^\theta)^{\nu_s}, \epsilon_s \xleftarrow{\$} \mathbb{Z}_p$
$H =$	$\mathsf{Hash}(\mathsf{hk}_s, \mathsf{vtk}, \mathsf{ek}, \mathfrak{L}_s, \mathcal{C})$	$\prod_i \left((e(\mathcal{C}_{i,s_i,1}/g^{s_i}, g_1)/e(f, a_i))^{\alpha_s} \mathcal{C}_{i,s_i,2}^{\beta_s} \mathcal{C}_{i,s_i,3}^{\mu_s} \mathcal{C}_{i,s_i,4}^{\nu_s}\right)^{\epsilon_s^i}$
$H' =$	$\mathsf{ProjHash}(\mathsf{hp}_s, \mathsf{ek}, \mathfrak{L}_s, \rho)$	$e(\prod_i \mathsf{hp}^{\rho_{i,s_i}\epsilon_s^i}, g_1)$

Fig. 14. Instantiating a 1-out-n Oblivious Transfer, SPHF is derived using the framework from [BBC+13a]

Generic Construction and Pairing-Based Instantiation. From this small change, we can give a generic construction which directly falls from our generic construction for OLBE. We consider, for every line i, the inner language $\mathfrak{L}_i$ of valid chameleon hashes of the line $i \in \{1,\ldots,k\}$ under the corresponding chameleon hash key ck and the verification key vk, given by the authority (see Section A page 35 for the definitions for chameleon hash). In this case, the language is not keyed (anyone can sample a word in it), but it possesses a trapdoor $\mathsf{tk}_{\mathfrak{L}} = \mathsf{tk}$ which is the trapdoor of the chameleon hash. This trapdoor enables

the simulator to send a chameleon hash which can correspond to any line s which is exactly what is required in our proof of generic OLBE (see Appendix E).

Indeed, as explained in Section 3, when dealing with 1-out-of-n Oblivious Transfer, one usually considers a database with various lines, so the naive approach to fit in our framework would have been to consider a set of languages being the set of numerals $\{\{1\}, \ldots, \{n\}\}$, expecting the receiver to commit to a word in one of these languages $\mathfrak{L}_i$, so that he recovers the corresponding line number i of the database. But a drawback with this approach is that in case of an adaptive corruption of the receiver, one would need to be able to equivocate the commitment to the word, which would require for instance a commitment at least extractable and equivocable (and even SPHF-friendly, see [ABB+13]), and not a simple $\mathsf{CCA}-2$ encryption as we propose in our generic construction of OLBE. This is the reason why we propose to consider the set of languages composed of valid representations of the corresponding integer, which allows more freedom in the constructions. Each line is now indexed by a whole language instead of a single number.

F.3 k-out-of-n Oblivious Transfer

Ideal Functionality for k-out-of-n Oblivious Transfer. From the generic framework of OLBE and its adaptation to 1-out-of-n Oblivious Transfer described right above in the previous section, one easily gets the ideal functionality of a k-out-of-n Oblivious Transfer (OT) protocol, by simply letting $n_{\max} = k \in [\![1; n]\!]$. This leads to the ideal functionality depicted in Figure 15 (rewritten in Simple UC).

The functionality $\mathcal{F}_{(k,n)\text{-OT}}$ is parametrized by a security parameter $\mathfrak{K}$ and a set of languages $(\mathfrak{L}_1, \ldots, \mathfrak{L}_n)$ along with the corresponding public verification algorithms $(\mathsf{Verify}_1, \ldots, \mathsf{Verify}_n)$. It interacts with an adversary $\mathscr{S}$ and a set of parties $\mathfrak{P}_1, \ldots, \mathfrak{P}_N$ via the following queries:

- **Upon receiving an input $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1, \ldots, P_n))$ from party $\mathfrak{P}_i$**, with $P_i \in \{0,1\}^{\mathfrak{K}}$: record the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P_1, \ldots, P_n))$ and reveal $(\mathtt{Send}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$. Ignore further $\mathtt{Send}$-message with the same ssid from $\mathfrak{P}_i$.
- **Upon receiving an input $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (W_i)_{i \in I})$ where $I \subset \{1, \ldots, n\}$ and $|I| = k$ from party $\mathfrak{P}_j$:** ignore the message if $(\mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (m_1, \ldots, m_n))$ is not recorded. Otherwise, reveal $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j)$ to the adversary $\mathscr{S}$, send $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, \mathfrak{P}_i, \mathfrak{P}_j, (P'_i)_{i \in I})$ to $\mathfrak{P}_j$ where $P'_i = P_i$ if $\mathsf{Verify}_i(W_i, \mathfrak{L}_i)$ returns 1, and $P'_i = \perp$ otherwise. Ignore further $\mathtt{Receive}$-message with the same ssid from $\mathfrak{P}_j$.

Fig. 15. Ideal Functionality for k-out-of-n Oblivious Transfer $\mathcal{F}_{(k,n)\text{-OT}}$

Generic Construction and Pairing-Based Instantiation. One can obtain a generic construction by simply applying the 1-out-of-n OT generic construction described above in parallel k times together with a NIZK made by the sender proving that all sets of messages are the same. This also leads to a pairing-based instantiation (while using Groth Sahai proof as NIZK for example [GS08]).

F.4 Conditioned Oblivious Transfer

Ideal Functionality for Conditioned Oblivious Transfer. Oblivious Transfer protocols allow the user to query a line in a database without restriction: The user automatically gets the line he asks for. While this has already found many applications, one may consider a slightly more complicated problem in which each line of the database has some additional access restriction (such as credentials, accreditation level, ...). For instance, lines numbers 2 to 10 of a database may be only accessible to the members of an organisation, while line number 1 may be only accessible to the presidents of several associations.

This problem has already been studied in various forms: Priced Oblivious Transfer (introduced in [AIR01]), which allows the user to access a given line if he has enough cash to pay (so if his balance is greater than the line price); Conditional Oblivious Transfer (formally introduced in [DOR99] but with few conditions about user privacy); Access Control Oblivious Transfer [CDN09], which aims at solving this issue for users with credentials.

Our framework allows to supersede all these primitives[14]: Thanks to our generalization of the Oblivious Transfer ideal functionality (using languages instead of simple numbers for the lines requested), the ideal functionality remains the same, since the languages already allow to add the credentials, signatures, prices, etc., required by those specific forms of OT.

Generic Constructions and Pairing-Based Instantiations. Our framework also provides a means to obtain tightly secure instantiations under classical (non-interactive, non q-type) assumptions. The required line restrictions need to be compatible with each other (if the word required for a line needs 2 elements to be committed, while another line requires words that can only be committed with 20 elements, there is no way to equivocate the first one to the second one, except if we adapt the language for the first line to have *dead* commitment terms so that all the lines require the same commitment size).

This could encompass a lot of different controls: For example, each line can be protected by a different password; a line can require a signature (credential) from a given authority on the line number, on the user identity, on a pseudonym; one can imagine more complicated policies like a required hamming weight, a scalar product, range, ... or any combination of those.

Giving a specific instantiation for all those cases is out of the scope of this article but for example, priced Oblivious Transfer requires to prove that the

[14] Although some of them (like Priced Oblivious Transfer) may require additional machinery to (privately) keep track of each user's balance over the course of many transactions.

	Generic 1-out-of-n Conditioned Oblivious Transfer
	Inner language
	Trapped Verifiable Commitment Keys $(\mathsf{ck}, \mathsf{tk})$
$c =$	$\mathsf{Commit}_{\mathcal{V}}(s)$
$\mathfrak{L}_s =$	$\{c \mid \mathsf{Verify}(c, s) = 1 \wedge c \in \{\text{Access Requirement s}\}\}$
$\mathsf{sk}_{\mathfrak{L}} =$	$\{\mathsf{sk}_s\}$
$\mathsf{tk}_{\mathfrak{L}} =$	$\mathsf{tk}_{\mathfrak{L}}$
	Compatible CCA Encryption
$\mathsf{ek} =$	ek
$\mathcal{C} =$	$\mathsf{Encrypt}(\mathsf{ek}, c; \rho)$
	SPHF
$\mathsf{hk}_s =$	$\mathsf{HashKG}(\mathsf{ek}, \mathfrak{L}_s)$
$\mathsf{hp}_s =$	$\mathsf{ProjKG}(\mathsf{hk}_s, \mathsf{ek}, \mathfrak{L}_s, \mathcal{C})$
$H =$	$\mathsf{Hash}(\mathsf{hk}_s, \mathsf{ek}, \mathfrak{L}_s, \mathcal{C})$
$H' =$	$\mathsf{ProjHash}(\mathsf{hp}_s, \mathsf{ek}, \mathfrak{L}_s, \rho)$

Fig. 16. Instantiating a 1-out-n Conditioned Oblivious Transfer

balance is greater than the line price[15]. The easiest way to do so would require to use an SPHF proving that the balance does not belong to the range $[\![0, \text{price}]\!]$. Both range proofs (adapting [BG13] with SPHF polynomial evaluations) and proof of negativity of a statement are already part of the SPHF toolbox.

[15] As already stated in Note 14, it also requires some additional machinery to (privately) keep track of each user's balance over the course of many transactions.

Table of Contents